VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky

DIPLOMOVÁ PRÁCE

2012 Miroslav Brtva

VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Vývoj BPStudia na platformě EMF

Development of BPStudio on EMF Platform

2012 Miroslav Brtva

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Miroslav Brtva**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Vývoj BPStudia na platformě EMF
Develpment of BPStudio on EMF Platform

Zásady pro vypracování:

Cílem této diplomové práce je připravit nástroj BP studio pro práci na platformě EMF.
1. Vytvořit metamodel BPM metody použité v BPStudiu.
2. Integrovat metamodel na platformu EMF.
3. Vyvinout funkční prototyp BPStudia na platformě EMF .
3. Vytvořit exportní a importní nástroj pro import a export modelů v jiných jazycích.

Seznam doporučené odborné literatury:

Podle pokynů vedoucího diplomové práce.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Svatopluk Štolfa, Ph.D.**

Datum zadání: 19.11.2010
Datum odevzdání: 04.05.2012

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA * OSTRAVA * Fakulta elektrotechniky a informatiky

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně.

Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne 29. června 2012 …..……...

podpis

Poděkování

Chtěl bych poděkovat mému vedoucímu diplomové práce Ing. Svatopluku Štolfovi, Ph.D. za odbornou pomoc a konzultace při realizaci této práce. Také bych chtěl poděkovat svému kolegovi Bc. Ondřeji Führerovi za úžasnou spolupráci při realizaci práce.

Abstrakt a klíčová slova

Abstrakt:

Tato práce je zaměřena na implementaci BP Studia a notací vhodných pro modelování byznys procesů na platformu EMF. Mezi notace patří EPC, BPMN a diagram aktivit. Samotná implementace zahrnuje vytvoření nástrojů pro modelování těchto notací. Teoretická část se zabývá popsáním teoretického základu pro implementaci práce, jedná se zejména o teorii metamodelování a EMF. V praktické části je rozvedeno použití EMF pro vytvoření metamodelů notací, které jsou základem implementace. Za využití GMF jsou tyto metamodely převedeny do pluginů, které umožňují graficky navrhovat byznys procesy v jednotlivých notacích. Praktická implementace popisuje jak architekturu implementace GMF a EMF obecně, tak pro konkrétní notace.

Klíčová slova:

Metamodeling, BP Studio, EMF, GMF, EPC, BPMN, diagram aktivit

Abstract:

This work is focused on the implementation of BP Studios and the appropriate notation for modeling business processes on the EMF platform. The notation is the EPC, BPMN and activity diagram. The actual implementation involves the creation of tools for modeling notation. The theoretical part deals with describing the theoretical basis for the implementation of the work, in particular the theory and EMF metamodelling. In the practical part of the elaborated use EMF to create a meta notation underlying implementation. GMF using these metamodels converted into plug-ins that allow you to graphically design business processes in different notations. Practical implementation describes implementation architecture of the EMF and GMF in general and for specific notation.

Key words:

Metamodeling, BP Studio, EMF, GMF, EPC, BPMN, Activity diagram

Seznam použitých symbolů a zkratek

EMF Eclipse Modeling Framework

GMF Graphical Modelling Framework

BPM Business Process Modeling

BPMN Business Process Modeling Notation

MOF Meta Object Facility

OMG Object Management Group

XML Extensible Markup Language

XMI XML Metadata Interchange

MDA Model Driven Architecture

UML Unified Modeling Language

API Application Programming Interface

EPC Event – Driven Process Chain

Obsah

1 Úvod

1.1 Stručné seznámení a cíl práce

Byznys procesy se dnes považují za nezbytnou součást strategie pro zlepšování konkurence schopnosti firem. Podniky se zaměřují na měření stávajících procesů, jejich porozumění a zlepšování, proto aby zákazník byl co nejvíce spokojen s poskytovaným produktem či službou. Existuje několik nástrojů pro modelování byznys procesů a v této diplomové práci se zabýváme několika z nich. Každá takto specifická notace je postavena na metamodelu, který definuje její objekty a chování, kterými poté můžeme modelovat byznys procesy. V rámci notací BP Studio, EPC, diagram aktivit a BPMN je cílem práce vývoj těchto modelovacích nástrojů na platformě Eclipse Modeling Framework. EMF je Java Framework a generátor kódu pro vytváření nástrojů a aplikací na základě strukturálního datového modelu. EMF pomáhá převést modely do efektního a snadno modifikovatelného Java kódu. Také je silným nástrojem pro definování metamodelů a je využit pro základní definici zmíněných notací. Tato práce je rozdělena do dvou praktických částí a je vypracována spolu s Bc. Ondřejem Führerem. V této diplomové práci a v její praktické části je popsán vývoj jednotlivých notací na platformě EMF a vytvoření pluginů, které umožňují jak grafický popis notací, tak i definování jejich elementů ve stromové struktuře. Součástí pluginů jsou i funkce a operace nad daným notacemi jako jsou transformace mezi jednotlivými notacemi a import/export do XML souborů. Jednotlivé transformace, import a export jsou součástí praktické části kolegy pana Führera a mým úkol je integrovat tyto součásti do pluginů.

1.2 Obsah kapitol

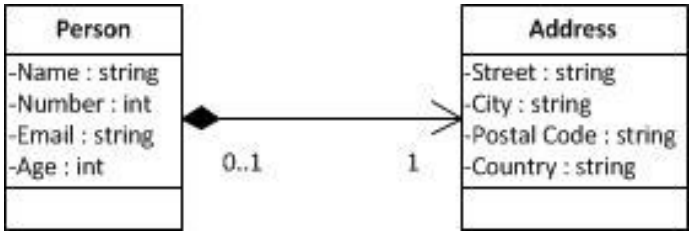
Diplomová práce je rozdělena do dvou hlavních částí, kde v první části, kapitoly 2 a 3, se zaměřuji na teoretickou část práce a popis jednotlivých preprekvizit pro vývoj notací na platformě EMF. Jedná se zejména o teorii metamodelování a základy EMF. Kapitola 4 je věnována praktické části implementace notací byznys procesů. Popisuje vývoj metamodelů na platformě EMF a další části potřebné pro vytvoření pluginů, které umožňují grafické modelování byznys procesů v prostředí Eclipse. Příloha A obsahuje část popisující instalaci a práci s pluginy jednotlivých notací, tato příloha je také součástí práce kolegy Ondřeje Führera, jelikož je manuálem pro instalaci a obsluhu námi implementované společné práce.

2 EMF

Eclipse Modeling Framework (EMF) je modelovací nástroj, který využívá možnosti poskytované platformou Eclipse. Model v EMF je méně obecný a neposkytuje příliš vysokou úroveň jako obecně uznávané interpretace. EMF nevyžaduje odlišné metodiky modelování nebo jakýkoliv sofistikovaný modelovací nástroj, jediné co je potřeba pro zahájení práce s EMF je Eclipse Java Development Tools. EMF zahrnuje modelovací koncepty přímo do své implementace, čímž zejména pro Java vývojáře, přináší výhody modelování do platformy Eclipse. EMF je Java framework a příslušenství pro generování kódu pro vytváření nástrojů a jiných aplikací založených na strukturovaném modelu. Snaží se poskytnout výhody formálního modelování s velmi nízkými vstupními náklady. Jeho cílem je pomoci při transformaci modelů do efektivního, správného a snadno přizpůsobitelného kódu v jazyce Java. Diagramy tříd, Diagramy spolupráce, Stavové diagramy, pro tyto diagramy je notace definována standardem UML a jeho kombinacemi může být specifikován kompletní model aplikace. Tento model může být použit čistě pro dokumentační účely nebo, v případě použití vhodných nástrojů, může být použit jako vstup, ze kterého se vygeneruje část nebo, v jednodušších případech, celá aplikace. Modely mohou být vytvářeny s použitím anotovaného Java kódu, XML dokumentů, nebo různými modelovacími nástroji. Tyto modely jsou následně importovány do EMF. EMF je tvořen dvěma základními frameworky: Core Framework a EMF.Edit. Core framework slouží jako základ pro generování implementačních tříd pro model v jazyku Java. EMF.Edit je postaven na základu Core Framework, rozšiřuje jej přidáním podpory pro vytváření editorů strukturovaných informací na základě formálně popsaného modelu. [5], [6], [7]

2.1 Základní specifikace EMF

Pro lepší pochopení co EMF zvládne a co jej reprezentuje, je potřeba definovat tři základní reprezentace, které EMF využívá. Jedná se o UML, XML a Java. Pokud se na tyto technologie zaměříme z pohledu vývoje aplikace. Každá z nich nám reprezentuje určitý datový model aplikace, tudíž reprezentuje ten samý model aplikace ze svého specifického pohledu. Pro UML využijeme nejčastěji třídní diagram a vytvoříme si základní model tříd pro specifikaci rozhraní. Jako příklad si uvedeme diagram popisující třídu Person a třídu Address.



Obr. 1: UML diagram tříd

Pomocí XML jsme schopni definovat stejný model, jako definuje model UML diagram tříd. Tento příklad reprezentace modelu, si můžeme tudíž popsat jako XML schéma, které nám bude definicí XML dokument.

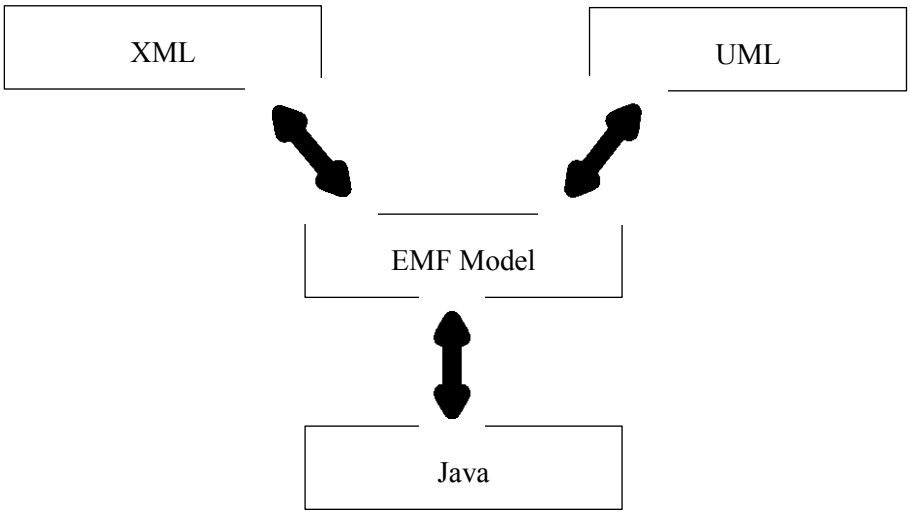
```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema>
```

```
<xsd:complexType name="Person">
        <xsd:sequence>
                <xsd:element name="Name" type="xsd:string"/>
                <xsd:element name="Number" type="xsd:int"/>
                <xsd:element name="Email" type="xsd:string"/>
                <xsd:element name="Age" type="xsd:int"/>
                <xsd:element name="lives at" type="po:Address" minOccurs="1"
                maxOccurs="1"/>
        </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="Address">
        <xsd:sequence>
                <xsd:element name="Street" type="xsd:string"/>
                <xsd:element name="City" type="xsd:string"/>
                <xsd:element name="Postal Code" type="xsd:string"/>
                <xsd:element name="Country" type="xsd:string"/>
                <xsd:element name="who lives" minOccurs="0" maxOccurs="1"/>
        </xsd:sequence>
</xsd:complexType>
</xsd:schema>
```

Poslední reprezentací, kterou jsme schopni popsat daný příklad je Java interface.

```
Public interface Person
{
        ...
}
Public interface Address
{
        ...
}
```

Za použití těchto reprezentací jsme si popsali stejný datový model aplikace. EMF je framework a nástroj pro generování kódu, který nám umožní za pomoci jedné z těchto reprezentací generovat reprezentace ostatní (UML diagram, XML Schéma, Java interface) a k tomuto také patřičné implementační třídy. EMF sjednocuje technologie XML, UML a Java díky EMF Modelu.



Obr. 2: Specifikace EMF

2.2 Definice Modelů

EMF je nejen framework, který slouží pro popis modelu a z něj generování dalších věcí, ale také je nástrojem, který je integrovaný a uzpůsobený pro efektivní programování. Tudíž pro EMF a jeho konceptuální model může modelování a programování znamenat tu samou věc. EMF stojí uprostřed modelování a programování a snaží se o správnou úroveň modelování při vývoji softwaru. Návrháři EMF přináší přiměřené množství modelování s programováním, aby vytvořili technologii, které je efektivní a přináší maximální efektivitu obou přístupů.

EMF Model je v podstatě podmnožinou třídního diagramu modelovacího jazyka UML, což je jednoduchý model tříd nebo dat aplikace. Mapování mezi EMF modelem a Javou je pro pochopení Java programátora jednoduché, stejně tak je i dostačující pro podporu detailních integračních dat mezi aplikacemi a spolu s výhodami EMF na zvýšení produktivity programátora při generování kódu, je to jedna z nejdůležitějších výhod modelování.

Modelování a programování z pohledu EMF může tedy znamenat tu samou věc. Jak jsme si již uvedli, EMF model z konceptuálního pohledu můžeme popsat několika různými způsoby: Java, UML nebo XML Schéma. Pro základní konceptuální model jednotlivých notací EMF jsme si v příkladu z kapitoly 2.1. definovali:

1. Person, Address: pro UML, Java jako definice tříd, XML schéma jako komplexní definici typu.
2. Name, Number, Street, City: mapují atributy v UML, dvojice get() a set() metody v Java jazyce a vnořené deklarace elementů v XML Schématu.
3. lives at, who lives: v UML zastupují asociace a reference, get() metodu v Javě a vnořené deklarace elementů odlišného typu než atributy v XML Schématu.

K definici EMF modelů za využití těchto částí jednotlivých modelů je zapotřebí metamodel, který je základním spojovacím prvkem. Tento model ponese informace o modelu, které jsou zapotřebí pro EMF nástroje a generátor. Tento metamodel se nazývá Ecore Model.

2.3 Ecore Model

Model využívaný pro reprezentaci modelů v EMF se nazývá Ecore, jež je EMF modelem, který má svůj metamodel. Proto Ecore Model je metametamodelem EMF. Ecore model můžeme vytvořit přímo nebo z Javy, UML, XML Schéma nebo z ostatních zdrojů, pro které nadefinujeme strukturu sérií instancí modelu. Tedy specifikujeme typy objektů, které vytvoří instance modelu a vztahy mezi nimi. Ecore metamodel definuje strukturu objektů v Ecore Modelu a je také sám sobě metamodelem. Ecore má své kořeny v MOF a UML a je navržen pro snadné mapování na Java implementace. Ecore také podporuje vyšší úrovně návrhů, které nejsou přímo zahrnuty v Javě, jako jsou omezení a obousměrné vazby. Proto je jednou z výhod EMF schopnost generovat korektní a efektivní Java implementace zahrnující i tyto omezení a vazby, pro urychlení práce programátorů.

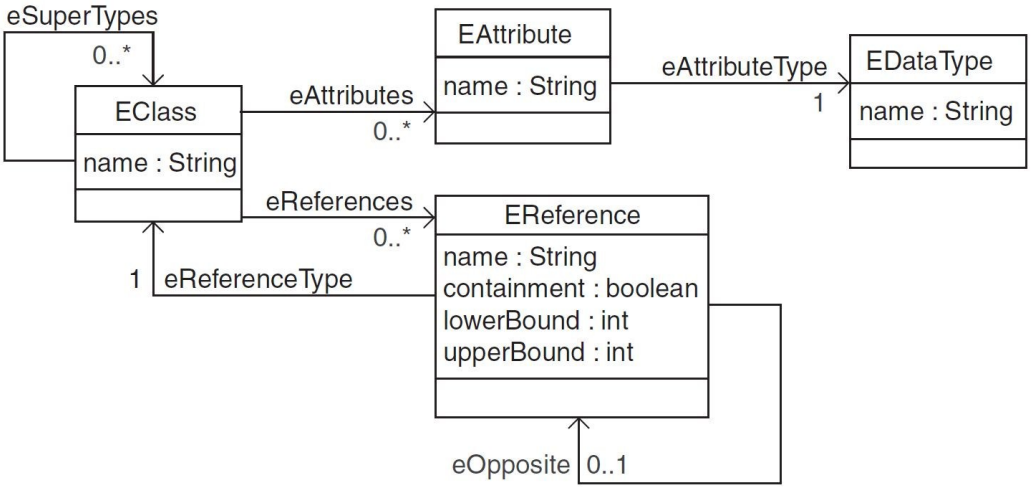
Jako jakákoliv instance v EMF, také Ecore Model může být sestaven programově nebo může být načten v serializované formě. Obecně se využívá ve dvou kontextech a to buď při vývoji aplikace, nebo při jejím běhu.

Během vývoje je Ecore Model hlavním zdrojem informací pro EMF generátor, který generuje kód aplikace. Kód zahrnuje interface a třídy realizující modelované typy, factory pro jejich konkretizaci, a package, který sestaví Ecore Model za běhu a poskytne přístup. EMF generátor čte Ecore Modely z jejich serializace XMI.

Za běhu aplikace, Ecore Model využívá framework k vymezení korektního chování pro daný model a také pro dynamické zjištění údajů o příslušném modelu. Chování všech objektů dynamického modelu kompletně závisí na Ecore Modelu. Také je možné za běhu Ecore model sestavit programově nebo načíst ze serializace, ale více efektivní je modely generovat. Následující kapitoly podrobně popisují Ecore metamodel.

2.3.1 Jádro Ecore

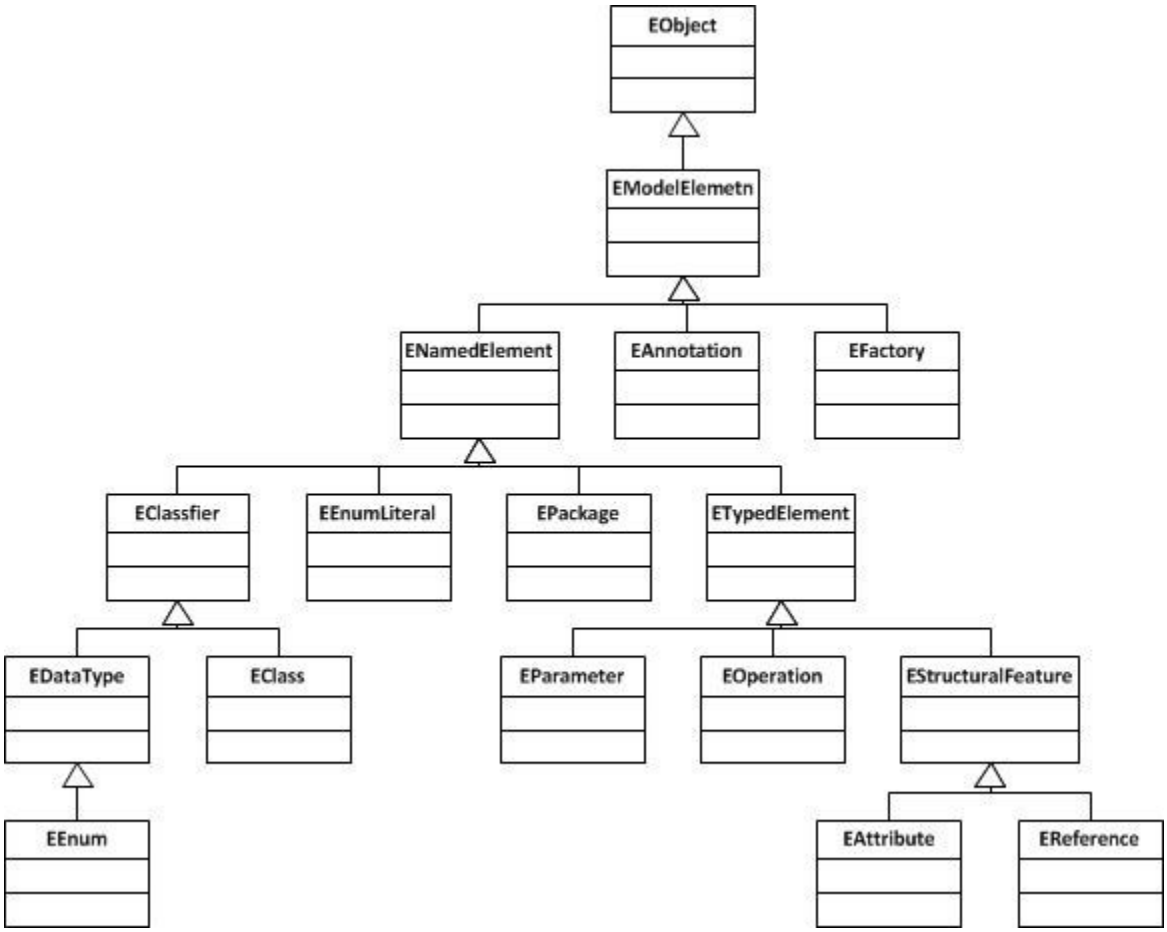
Pro definici struktury Ecore Modelu a jeho metamodelu si popíšeme základní elementy modelu. Tyto elementy jsou použity při vytvoření metamodelů notací implementace. Podmnožina Ecore Modelu se skládá ze čtyř objektů:



Obr. 3: Podmnožina Ecore Modelu

1. **EClass** reprezentují v modelu třídy. Třídy jsou identifikovány jménem a mohou mít nespočet atributů a vazeb. V rámci dědičnosti se mohou odkazovat na různé třídy
2. **EAttribute** se využívá k reprezentaci modelovaných atributů, jsou identifikovány jménem a typem.
3. **EDataType** reprezentují jednoduché typy, které nejsou definovány jako třídy. Mohou to být jednoduché typy jako int nebo float, nebo typy objektů definované v Javě jako java.util.Date. Datové typy jsou identifikovány jménem a využívají se jako typy atributů.
4. **EReference** je využita k modelování asociací mezi třídami a specifikuje jeden konec asociace. Pokud je vazba obousměrná je potřeba referenci přiřadit také spodní a horní hranice četnosti. Jsou identifikovány jménem a typem.

Definovaná podmnožina Ecore Modelu popisuje třídy **EClass** a jejich atributy, které jsou modelovány za pomoci **EAttribute** a vazeb modelovaných s **EReference**. Tato podmnožina je součástí metamodelu Ecore Modelu a definuje vztah mezi nimi, je to jen malá část kompletního metamodelu. V dalších kapitolách si podrobně rozebereme jednotlivé části metamodelu Ecore modelu.



Obr. 4: Metamodel EMF

2.3.2 Strukturální vlastnosti – EstructuralFeature

Třídy **EAttribute** a **EReference** jsou pro svou podobnost zařazeny pod **EStruturalFeature**, jež je jejich společným základem. Obě tyto třídy jsou definovány jménem, typem, obě definují instanci třídy **EClass**, která je obsahuje, a také obsahují stejné atributy **lowerBound** a **upperBound** pro definici spodní a horní hranice omezení. **EStructuralFeature** je odvozená také z **ENamedElement** a **ETypedElement**.

ENamedElement definuje jediný atribut **name**(jméno), který dědí většina tříd Ecore Modelu.

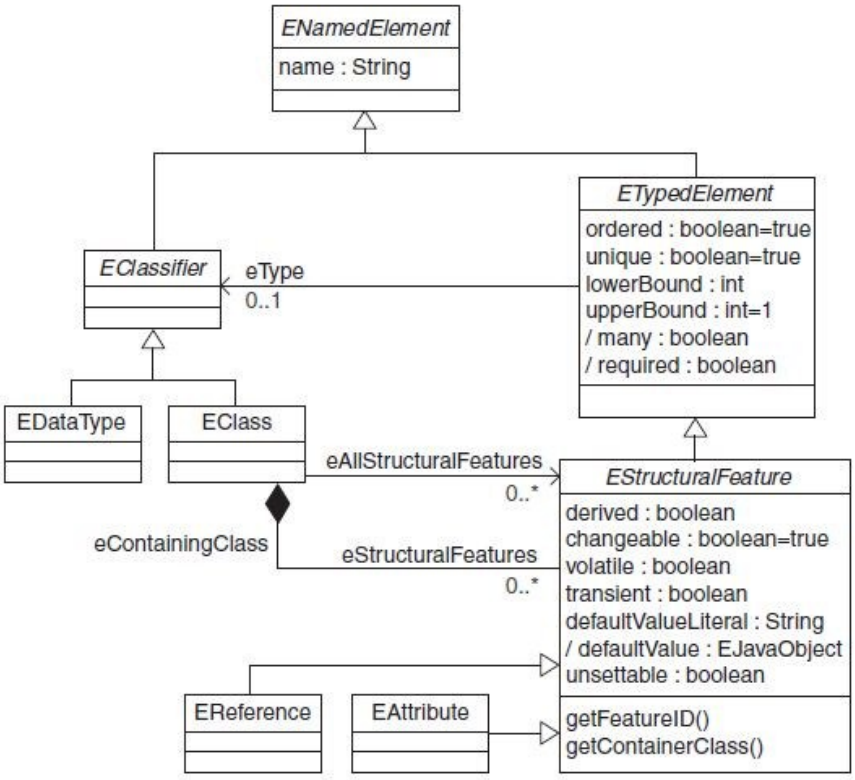
ETypedElement definuje atributy spojené s mnohočetností, jedná se jak o atributy typu hodnot, ale také jaký počet je těchto hodnot povolen. Atributy **lowerBound** a **upperBound** definují tyto hodnoty a mohou nabývat hodnot od 0 po kladný integer. Hodnota **upperBound** až po hodnotu neomezeno (značeno *). Další atributy vyznačují typy mnohočetnosti, jsou to atributy **unique** a **ordered**. Atribut **unique** určuje, aby se daná hodnota nevyskytovala více než jednou. Atribut **ordered** specifikuje významnost pořadí hodnot. **ETypedElement** obsahuje také referenci **eType**, jež spojuje třídu **EClassifier**, z důvodu zachycení požadovaných typů třídy vycházejících z třídy **EClassifier** (**EDataType** a **EClass)**.

Proto, aby byly atributy z **EStructuralFeature** dostupné v třídě **EClass,** využívá referenci **eStructuralFeatures**. Tato reference je obousměrná, tudíž třída **EClass** je obsažena také v **EStructuralFeature** vazbou **eContainingClass**.

Třída **EStructuralFeature** definuje několik boolean atributů společných pro **EAttribute** a **EReference**. Jsou to atributy **changeable**, **derived**, **transient**, **unsettable**, **volatile**, které definují ukládání a přístup k hodnotám, a atributy **defaultValueLiteral** a **defaultValue**, které definují výchozí hodnoty před jejich nastavením.

- **changeable** rozhoduje, zda je možné nastavit hodnotu vlastnosti externě
- **derived** specifikuje, zda hodnota vlastnosti může být vypočtena ze souvisejících dat
- **transient** určuje, zda daná vlastnost je vynechána ze serializace objektu, ke kterému přísluší
- **unsettable** specifikuje, zda daná vlastnost již byla nastavena (unset)
- **volatile** specifikuje, zda daná vlastnost má přímo asociované uložiště, využívá se pro odvozené vlastnosti
- **defaultValueLiteral** je definováno jako typ string
- **defaultValue** zpřístupní typy ekvivalentní hodnot jako u Java objektu

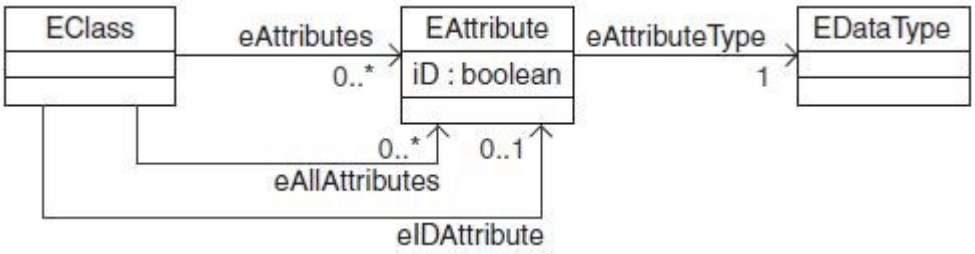
V třídě **EStructuralFeature** jsou definovány také dvě operace. Metoda **getFeatureID** zajišťuje nastavení jednoznačného identifikátoru vlastnosti při inicializaci statického modelu a během volání metody getEAllStructuralFeatures() při dynamickém definování třídy. Metoda **getContainerClass,** využívaná při generování modelu, spojuje příslušné třídy, ze kterých je definovaná či zděděná.



Obr. 5: Ecore – strukturální vlastnosti

2.3.3 Atributy – EAttribute

EAttribute definuje jeden atribut ID ve své třídě, který určuje, zda hodnota atributu, může být použita k jednoznačné identifikaci instance třídy. Také definuje odvozenou referenci **eAttributeType** k určení typu. Třída **EClass** definuje tří relace propojující **EAttribute**, jsou to **eAttribute**, **eAllAttributes** a **EIDAttribute**, které slouží k získávání informací o příslušných atributech a ID atributů.



Obr. 6: EAttribute

2.3.4 Reference – EReference

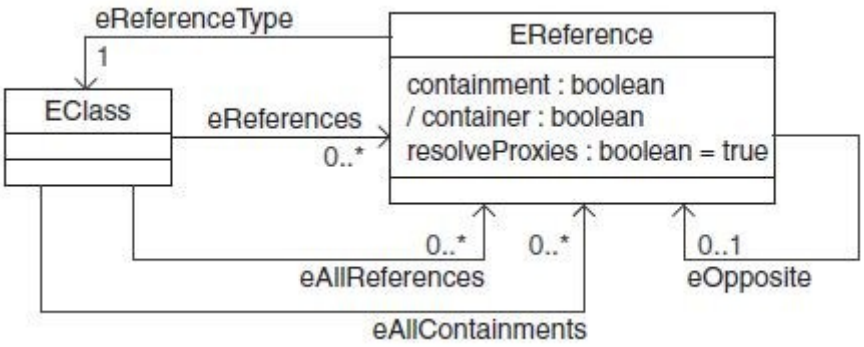
Implementace reference **EReference** v metamodelu Ecore využívá tří atributů a dvou referencí. Mezi atributy patří:

- **containment** – nese informaci o tom, zda daná **EReference** je využita v modelování
- **container –** odvozený atribut, který definuje, zda daná reference je obousměrná
- **resolveProxies** – udržuje perzistenci zdroje modelu

Reference:

- **eReferenceType –** spojeju třídu **EClass**, z důvodu zachycení požadovaných typů třídy
- **eOpposite –** reference reprezentuje obousměrnou asociaci vazby **EReference**

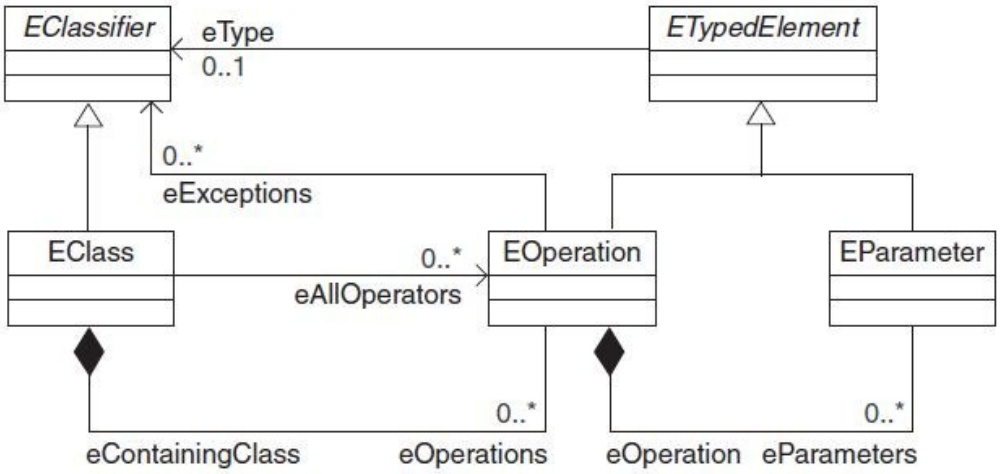
Třída **EClass** poskytuje odvozené asociace pro **eReferences**(poskytuje reference třídě **EClass**), **eAllReference**(poskytuje všechny reference) a **eAllContainments**(poskytuje reference, které mají hodnotu atributu **containment** *true*).



Obr. 7: EReference

2.3.5 Vlastnosti chování – EOperation

Ecore Model může zachycovat vlastnosti chování třídy **EClass** pomocí třídy **EOperation**, přesněji řečeno zachycuje interface třídy, k tomu slouží asociativní vazby **eAllOperators** a obousměrná vazba **eContainingClass**, **eOperations**. **EOperation** obsahuje nula nebo více **EParameter**, tuto vazbu zajišťuje obousměrná asociace **eParameters** a **eOperation**. Poslední reference v této části modelu je **eExceptions**, modeluje typy objektů, jež operace může zachytit výjimku.

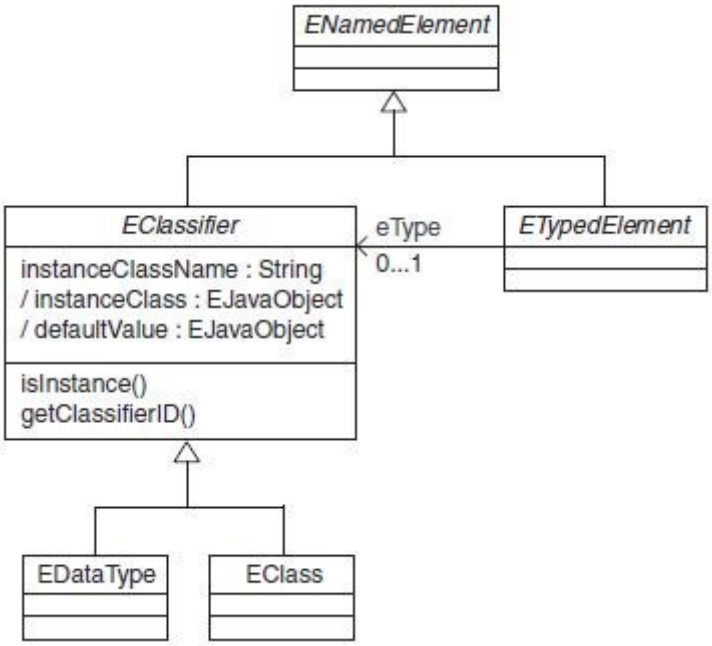


Obr. 8: EOperation

2.3.6 Klasifikátory – EClassifier

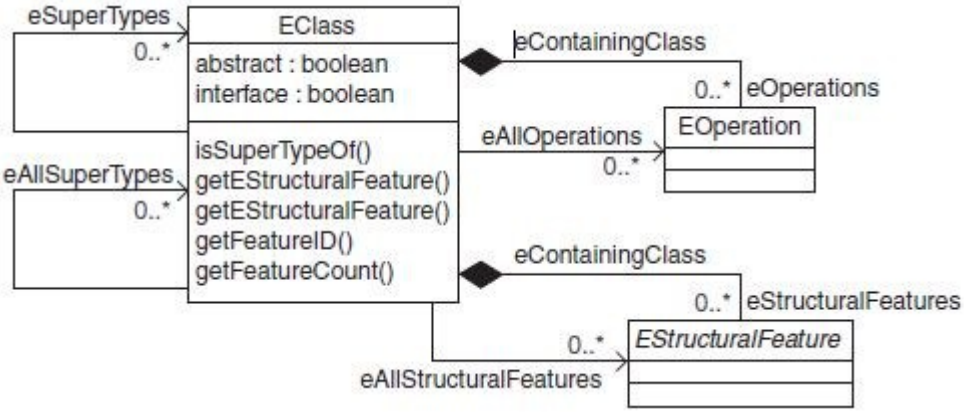
EClassifier je základní třídou pro třídy **EClass** a **EDataType**. Definuje tři atributy, první z nich **instanceClassName** specifikuje jménem třídu nebo interface. Další odvozený atribut **instanceClass** může být použit k přístupu k Java třídě jménem specifikovaným v atributu **instaceClaasName**. Poslední atribut odvozený z atributu **instanceClass** je **defaultValue**. Tento atribut reprezentuje skutečnou výchozí hodnotu třídy nebo datového typu. Operace **isInstance**, definovaná třídou **EClassifier**, testuje libovolné Java objekty specifikované jako parametry, zda jsou nenulové a instancí třídy nebo datového typu reprezentující **EClassifier**. Poslední specifikovaná

operace **getClassifierID** vrací integer hodnotu, jež je unikátním identifikátorem klasifikátoru **EClassifier**.



Obr. 9: EClassifier

2.3.7 Třídy – EClass



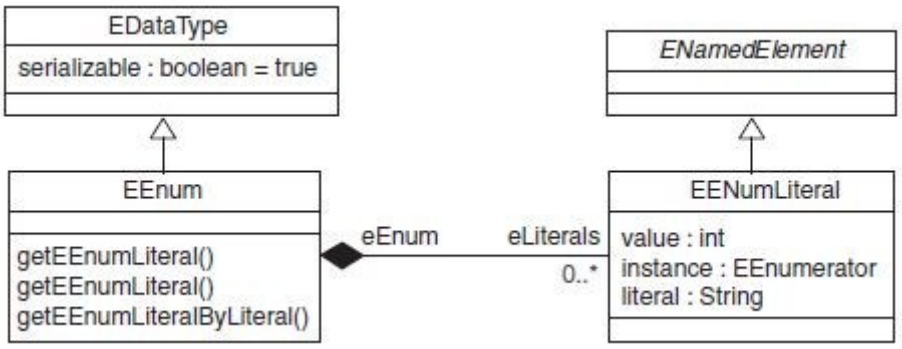
Obr. 10: EClass

Třída **EClass** definuje dva atributy, pokud je atribut **abstract** *true*, **EClass** reprezentuje abstraktní třídu a pokud je hodnota druhého atributu **interface** *true*, reprezentuje interface. Mezi abstraktní třídy patří **ENamedElement, EClassfier, ETypedElement, EStructuralFeature** a **EModelElement**. **EClass** definuje pět operaci. Operace **isSuperTypeOf** testuje, zda jedna třída rozšiřuje další, za pomoci reference **eAllSuperTypes**. Dvě metody **getEStructuralFeature** vrací hodnotu jméno nebo ID vlastnosti třídy **EStructuralFeature** přes referenc **eAllStructuralFeatures**. Operace **getFeatureID** vrací hodnotu ID vlastnosti a **getFeatureCount** vrací součet strukturálních vlastností definovaných a děděných třídou. Reference **eContainingClass, eOperations** a

eContainingClass, eStructuralFeatures zachycuje vzájemný vztah **EClass** s **EOperation** respektive s **EStructuralFeature**. Poslední referencí je **eSuperTypes**, která poskytuje podporu pro několikanásobnou dědičnost třídě **EClass.**

2.3.8 Datové typy – EDataType

Datové třídy v modelu Ecore reprezentují jednu část dat. **EDataType** modeluje Java typ, Java třídu, základní typ, interface nebo pole. Datový typ definuje jen jeden atribut **serializable**, který indikuje, zda hodnoty typů mohou být serializovány.



Obr. 11: EDataType

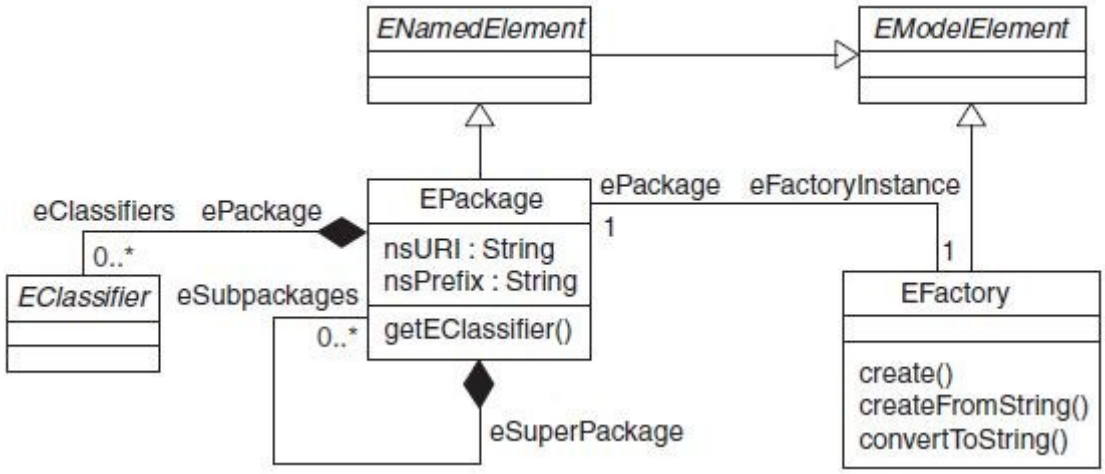
Enumerated Type (výčtový typ) je speciální datový typ, který definuje explicitní seznam hodnot nazývaný literály. **EDataType, EEnum** a **EEnumLiteral** modelují datové typy, výčtové typy a literály. **EEnum** specifikuje **EEnumLiteral** přes obousměrnou vazbu **eLiterals, eEnum**. **EEnumLiteral** definuje atributy **value** a **literal**, které reprezentují literál. Hodnoty literálu jsou k dispozici jako statické instance a jsou dostupné přes atribut **instance**. **EEnum** má tři operace, které vrací **EEnumLiteral**. Operace **getEEnumLiteral** vyhledává hodnotu **name**(jméno) a **value**, a **getEEnmLiteralByLiteral** řetězec literálu.

2.3.9 Třídy Ecore Modelu Package a Factory – EPackage, EFactory

V Ecore Modelu jsou související třídy a datové typy seskupeny do takzvaných package (**EPackage**). Factory (**EFactory**) se využívá k vytváření instancí tříd a hodnot datových typů, které přísluší danému package. Při serializaci Ecore Modelu reprezentuje package kořenový element (root element).

EPackage definuje atributy **nsURI**, který unikátně identifikuje package. Díky vztahu EMF s XML tento atribut je také použit k identifikaci XML namespace. Další atribut **nsPrefix** specifikuje patřičný prefix namespace. Metoda **getEClassifier** slouží k získání třídy **EClassifier**, toto je zajištěno obousměrnou referencí **eClassifier**, **ePackage**. **EPackage** podporuje vnoření za pomoci uzavřené reference **eSubpackages**. Poslední reference **eFactoryInstance, ePackage** sdružuje třídy **EPackage** a **EFactory**.

EFactory definuje jediné tři metody **create, createFromString, convertToString**, které vytváří instanci tříd a převádí hodnoty datových typů z a do řetězců.

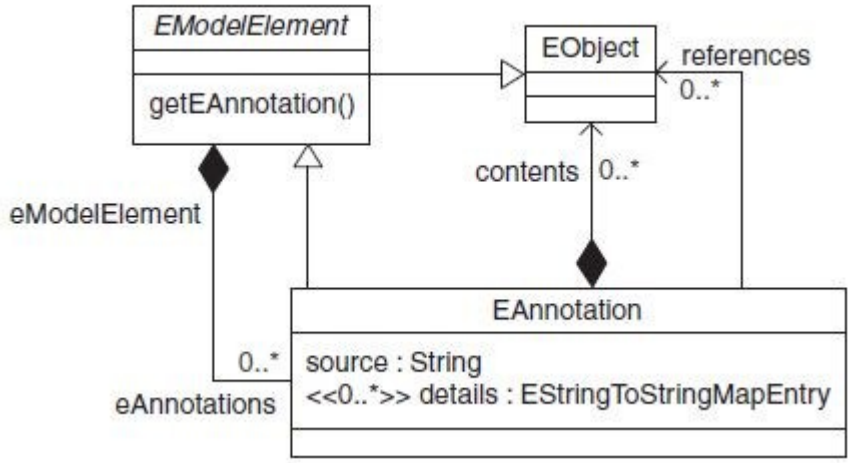


Obr. 12: EPackage a EFactory

2.3.10 Třída Ecore Model Annotation – EAnnotation

Třída anotace (**EAnnotation**) představuje mechanizmus, kterým lze přidávat informace k objektům z Ecore Modelu. **EAnnotation** je podtřídou **EModelElement** a díky vazbě **eAnnotations, eModelElement** má přístup ke všem třídám definovaných v Ecore Modelu. Třída anotace obsahuje také atribut **source**, který se využívá k uložení identifikátoru reprezentující typ anotace, díky tomuto atributu operace **getAnnotation** z **EModelElement** přistupuje k **EAnnotation**. Poslední reference **contents** a **references** dovolují **EAnnotation** obsahovat nebo se odkazovat na libovolné EMF objekty. Atribut **details** zachycuje vlastnosti anotace pro speciální chování generátoru kódu.

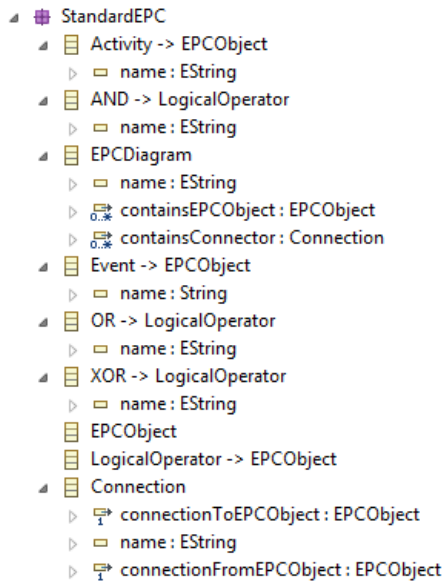
Anotace Ecore modelu se rozdělují do několika typů a využívá se k uložení informací jak pro generátor kódu, tak patřičných informací, které přebírají z modelů, z nichž se Ecore Model vytváří.



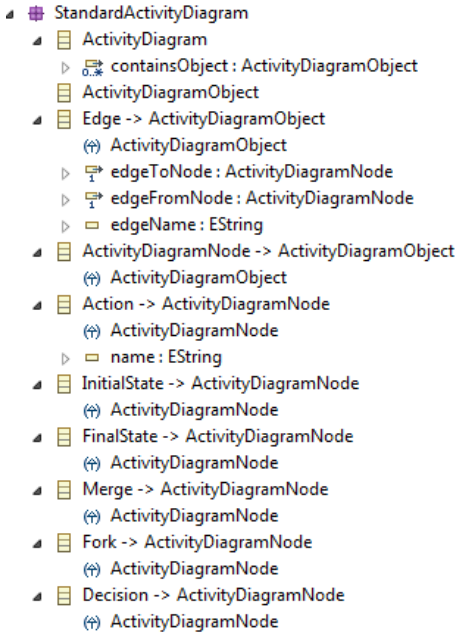
Obr. 13: EAnnotation

2.3.11 Ecore modely notací

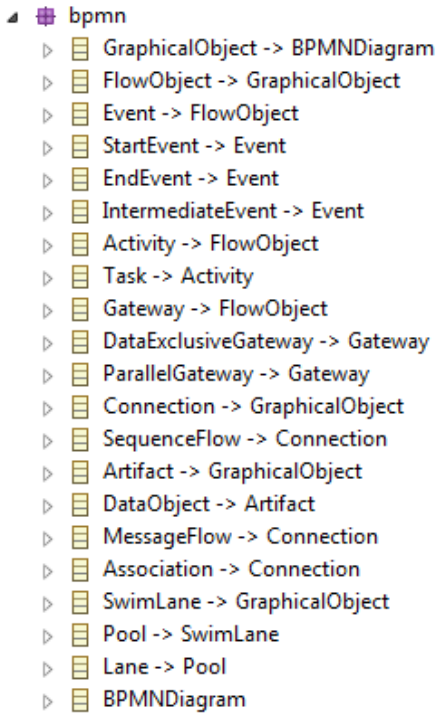
V rámci implementace prvotní krok byl definovat metamodely jednotlivých notací na platformě EMF. Jednotlivé metamodely jsou popsány v praktické části diplomové práce. Nyní si vyspecifikujeme Ecore modely notací, které byly vytvořeny. Ecore model je základem celé implementace na rozdíl od XML Schématu nebo UML modelu, XML a UML jsou v rámci práce vygenerovány z definovaných Ecore metamodelů notací. Pro svou komplexnost byl zvolen Ecore model, který je základním stavebním kamenem celé implementace, více je popsáno v praktické části.



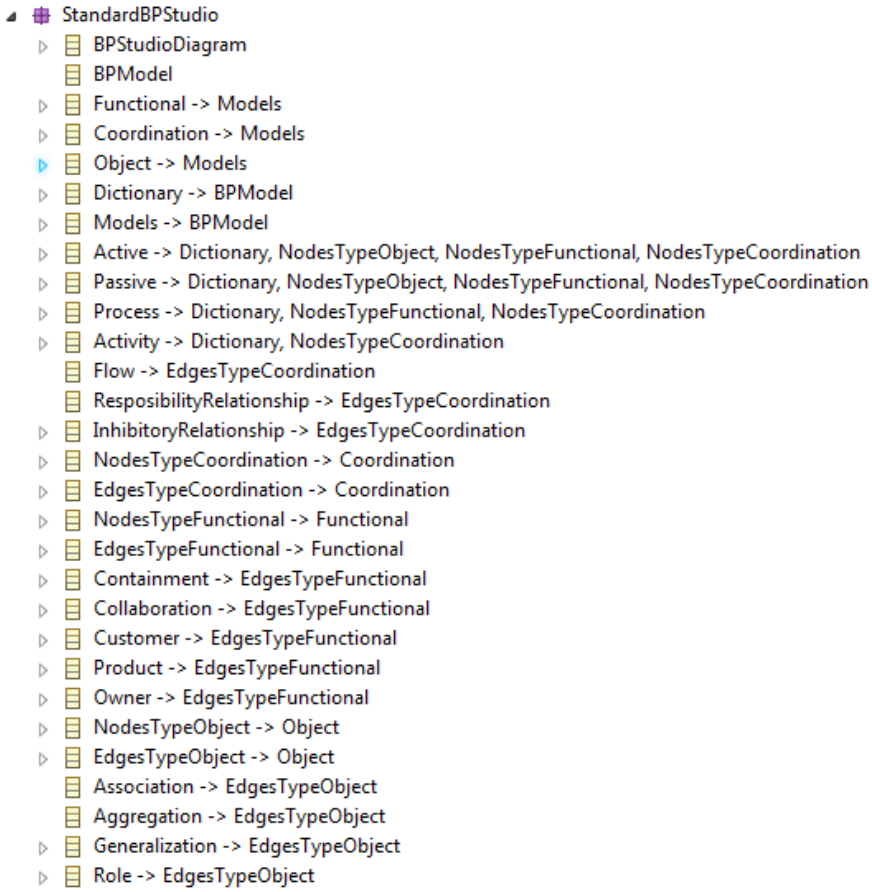
Obr. 14:EPC Ecore model



Obr. 15: Diagram Aktivit Ecore model



Obr. 16: BPMN Ecore model



Obr. 17: BP Studio Ecore model

2.4 XML Schéma

Pro vytvoření objektového modelu pro manipulaci struktury XML dat, EMF poskytuje vhodný přístup založený na XML Schématu [6]. EMF může vytvořit Ecore model, který koresponduje XML schématu. Ze schématu je ovlivněno generování kódu a EMF poskytuje Java API pro manipulaci instancí schématu. Výhodou využívání XML Schématu definovaného modelu je, že při serializaci budou instance modelu odpovídat XML Schématu. V základní úrovni je mapování XML Schématu do Ecore model poměrně jednoduché:

- Schéma je mapováno do **EPackage**.
- Complex Type je mapován do **EClass**.
- Simple Type definice je mapována do **EDataType**.
- Atribut nebo element deklarace jsou mapovány do **EAttribute**, jestliže jejich typ mapuje **EDataType**, nebo jsou mapovány do **EReference**, jestliže jejich typ mapuje **EClass**. K dispozici je speciální **EClass** nazývaná *DocumentRoot*, která drží globální elementy a atributy, jež nejsou vnořené do komplexního typu (complex type).

Ecore namespace atributům lze přizpůsobit výchozí mapování do Ecore modelu takto:

```
<?xml version="1.0" encoding="UTF-8"?>

 <xsd:element name="tree" type="tree:Node"/>

 <xsd:complexType name="Node">

        <xsd:sequence>

         <xsd:element ecore:opposite="parent"

                maxOccurs="unbounded" minOccurs="0" name="children"

               type="tree:Node"/>

       </xsd:sequence>

       <xsd:attribute name="label" type="xsd:string"/>

 </xsd:complexType>

</xsd:schema>
```

2.4.1 XML specifikace pro Ecore Model

Vlastnost	Použitelnost	Hodnota	Použití
ecore: changable	<xsd:element> <xsd:attribute>	true | false	**changeable** atribute z **EStructuralFeature** je nastaven na specifikovanou hodnotu.
ecore: constraints	<xsd:complexType> <xsd:simpleType>	List-of-Names	**EStructuralFeature** je komentováno zdrojovým Ecore **EAnnotation**, která obsahuje **key** s hodnotou‘constraint’ tato hodnota je seznam s názvy omezení hodnot oddělených mezerou.
ecore: default	<xsd:element> <xsd:attribute>	Literalvalue	Odvozená hodnota literálu atributu **EAttribute** je nastavena na specifickou hodnotu.
ecore: derived	<xsd:element> <xsd:attribute>	true | false	Atribut **derived** **EStructuralFeature** je nastaven na specifickou hodnotu.
ecore: documentRoot	<xsd:schema>	Name	Document root název třídy **EClass** je nastaven na specifickou hodnotu.
ecore: enum	<xsd:simpleType>	true | false	Pokud je *false*, pak mapování simple type s výčtem aspektů **EEnum**, mapuje **EDataType** s výčtem rozšířených komentářů meta dat.
ecore: featureMap	<xsd:element><xsd:sequence> <xsd:schoice><xsd:all>	Name	Pokud je specifikován prázdný řetězec, Ruší mapování do featureMap. Pokud je specifikován, hodnota **name** z featureMap je **EstructuralFeature**.
ecore: ignore	<xsd:attribute> <xsd:element> <xsd:annotation> XSD facets	false | true	Pokud je *true* na všech elementech nebo atributech, komentáře nebo jejich dokumentace nebo obsahuje appinfo, nebo simple type aspekty, specifikuje, že příslušný Ecore návrh nebude produkován normálně.
ecore: implements	<xsd:complexType>	List-of-QName	Specifikuje další *super types*, označením QNames pro **EClass**.
ecore: instance Class	<xsd:complexType> <xsd:simpleType>	Java-Class	Specifikuje název instance typu **EClassifier**; ‘{}’ může být využito místo ‘<>’.
ecore: interface	<xsd:complexType>	true | false	**interface** atribut z **EClass** je nastaven na specifickou hodnotu.

ecore: key	<xsd:appinfo>	Stringvalue	Specifikuje **key** z detailního vstupu **EAnnotation** kde obsah **Appinfo** jsou hodnoty.
ecore: lowerBound	<xsd:attribute> <xsd:element>	Integervalue	Specifikuje hodnotu **lowerBound** atributu z **EStructuralFeature**.
ecore: many	<xsd:attribute>	true | false	Pokud je *true,* je použit na atribut s list-type hodnotou, toto zaručí mapování mnohoznačnosti mnoha vlastností příslušných na typy položek.
ecore: mixed	<xsd:complexType>	true | false	complex type je mapován Přesně tak, jako by měl skutečný smíšený atribut se zadanou hodnotou.
ecore: name	XSD named components	Name	Specifikuje název **name** **ENamedElement**.
ecore: nsPrefix	<xsd:schema>	NCName	Specifikuje **nsPrefix** z **EPackage**.
ecore: opposite	<xsd:attribute> <xsd:element>	Name	Specifikuje název elementu nebo atributu v **EReference** typu, Který bude spárován jako jeho protiklad. Pokud je využita uzavřená reference, container reference bude vytvořena.
ecore: ordered	<xsd:attribute> <xsd:element>	true | false	Nastavuje hodnotu **ordered** atributu na **EStructuralFeature**.
ecore: package	<xsd:schema>	Name	Specifikuje název Java package, poslední segment, který bude využit jako název **EPackage**.
ecore: reference	<xsd:attribute> <xsd:element>	QName	Pokud je využit atribut nebo element typu IDREF, IDREFS, nebo anyURI, QName specifikuje complex type příslušný k **EClass**, která je typem **EReference**.
ecore: resolve Proxies	<xsd:attribute> <xsd:element>	true | false	Specifikuje **resolveProxies** atribut z **EReference**.
ecore: serializable	<xsd:simpleType>	true | false	Specifikuje **serializable** atribut z **EDataType**.
ecore: suppressed Get Visibility	<xsd:attribute> <xsd:element>	true : false	Pokud je *true*, **EStructuralFeature** je okomentováno s GenModelsourced **EAnnotation**, která potlačuje přistupující get() pro

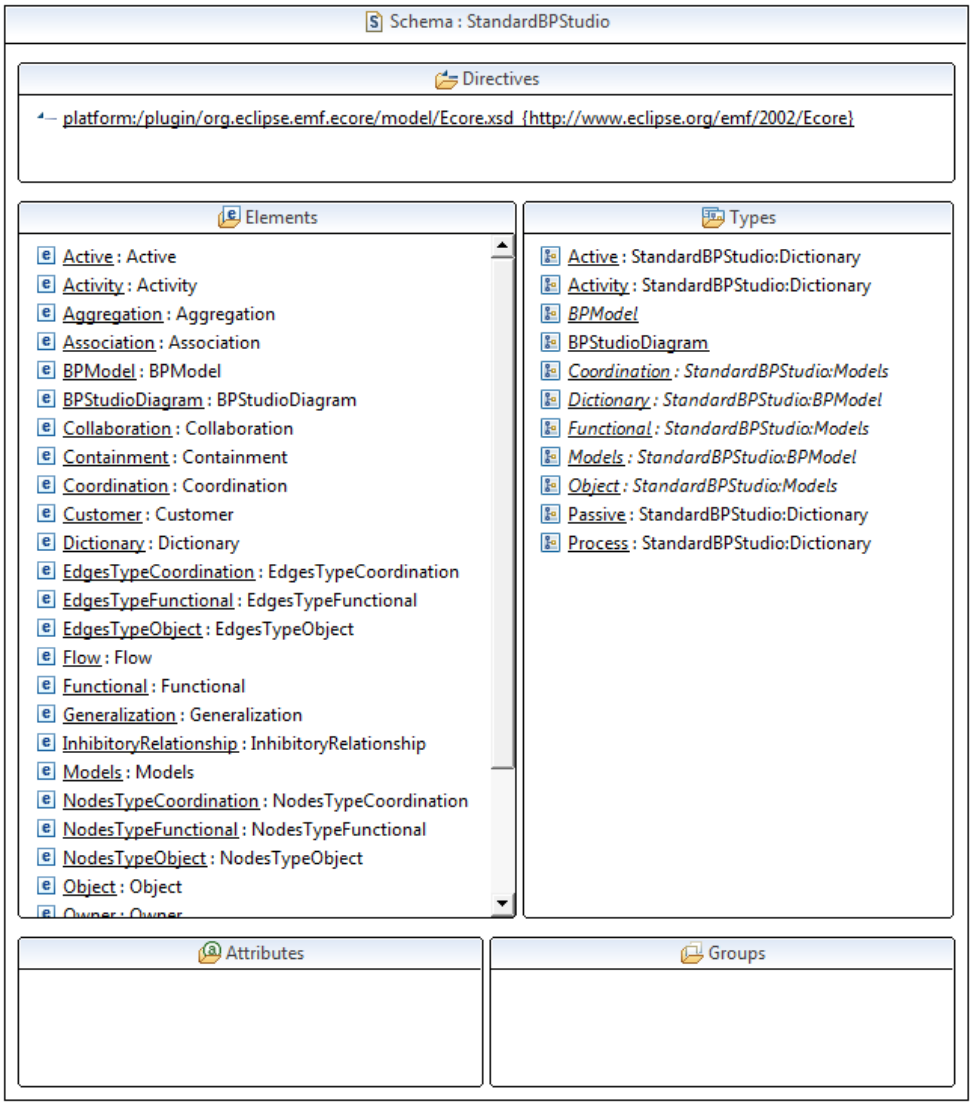
			vlastnost v interface.
ecore: suppressed IsSet Visibility	<xsd:attribute> <xsd:element>	true : false	Pokud je *true*, **EStructuralFeature** je okomentováno s GenModelsourced **EAnnotation**, která potlačuje přistupující isSet() pro vlastnost v interface.
ecore: suppressed Set Visibility	<xsd:attribute> <xsd:element>	true : false	Pokud je *true*, **EStructuralFeature** je okomentováno s GenModelsourced **EAnnotation**, která potlačuje přistupující set() pro vlastnost v interface.
ecore: suppressed Unset Visibility	<xsd:attribute> <xsd:element>	true : false	Pokud je *true*, **EStructuralFeature** je okomentváno s GenModelsourced **EAnnotation**, která potlačuje přistupující unset() pro vlastnost v interface.
ecore: transient	<xsd:attribute> <xsd:element>	true : false	Specifikuje **transient** atribut z **EStructuralFeature**.
ecore: unique	<xsd:attribute> <xsd:element>	true : false	Specifikuje **unique** atribut z **EStructuralFeature**.
ecore: unsettable	<xsd:attribute> <xsd:element>	true : false	Specifikuje **unsettable** atribut z **EStructuralFeature**.
ecore: upperBound	<xsd:attribute> <xsd:element>	Integer	Specifikuje **upperBound** atribut z **EStructuralFeature**.
ecore: value	<xsd:enumeration>	Integer	Specifikuje **value** atribut z **EEnumLiteral**.
ecore: volatile	<xsd:attribute	true : false	Specifikuje **volatile** atribut z **EStructuralFeature**.

Tab. 1: Specifikace XML pro Ecore Model[1]

2.4.2 XML Schéma implementace

EMF umožňuje definovat xsd dokumenty pomocí grafického editoru nebo přímo psaním xsd dokumentu. EMF je silným nástrojem, protože takto definované XML Schéma je možné přímo převést do Ecore modelu nebo také přímo z Ecore modelu vytvořit XML Schéma. Při implementaci je využito generování XML Schématu z Ecore modelu. Výše popsaná XML specifikace je v rámci implementace vygenerována z Ecore metamodelů jednotlivých notací. Ve zdrojových kódech každé notace jsou tyto XML Schémata uloženy ve složce „model“. Na následujících obrázcích je uveden příklad grafického editoru xsd dokumentu pro metamodel BPStudia.

[1] Zdroj http://refcardz.dzone.com/refcardz/essential-emf



Obr. 18: XML Schéma metamodelu BPStudia

2.5 Java Anotace

EMF dovoluje popis modelu také v Java kódu [6], tedy Ecore modelu jež je tvořen jednou nebo více **EPackage**, každá z nich obsahuje několik **EClass**, které jsou definovány několika **EAttribute**, **EReference** a **EOperation**. Každá **EPackage** také obsahuje několik **EDataType** a **EEnum**. Pokud budeme specifikovat model pomocí zdrojového kódu Java, třídy **EClass** jsou uvedeny jako Java interface a EAttribute, EReference a EOperation jsou odvozeny z metod jednotlivých interface. EEnum jsou specifikovány jako Java třídy. **EPackage** může být také specifikován jako Java interface, i když ve většině případů nemusí být uvedeno explicitně, protože mohou být definovány z použitých **EClass**. Obdobně **EDataType** může být definován explicitně, i když je většinou odvozen z příslušného typu Java. EMF vyžaduje komentování jednotlivých interface speciálně formátovaným komentářem, který identifikuje jednotlivé elementy modelu, a může poskytovat dodatečné informace, které nejsou přímo možné vyjádřit v kódu Java. Jedná se tedy o formu komentářů Javadoc, které předcházejí třídu, interface nebo definici metody, ke které se vztahují. Ty jsou označeny tagem *@model* a mají následující syntaxi:

```
/**
 * @model
 */
public interface Node {
 /**
  * @model opposite="parent" containment="true"
  */
 List<Node> getChildren();
 /**
  * @model opposite="children"
  */
 Node getParent();
}
```

2.5.1 Java specifikace pro Ecore Model

Java specifikace[2] pro Ecore Model popisuje jednotlivé třídy a jejich vlastnosti.

@model Vlastnosti pro Třídy

Java specifikace pro třídu **EClass** je Java interface, jež předchází *@model* tag.

Vlastnost	Hodnota	Použití
abstract	true | false	**abstract** atribut z **EClass** je nastaven na specifickou hodnotu.
interface	true | false	**interface** atribut z **EClass** je nastaven na specifickou hodnotu.

Tab. 2: Vlastnosti pro Třídy

@model Vlastnosti pro Typed Element

ETypedElement je specifikován metodou, jež je součástí interface příslušného **EClass**, které obsahuje **Typed Element**.

[2] Zdroj http://refcardz.dzone.com/refcardz/essential-emf

Vlastnost	Hodnota	Použití
lower or lowerBound	Integer-value	**lowerBound** atribut z **ETypedElement** je nastaven na Integer-value, která musí být 0 nebo větší.
many	true |false	Pokud je *true*, **upperBound** atribut z **ETypedElement** je nastaven na -1 (unbounded). Jinak je nastaven na 1.
ordered	true |false	**ordered** atribut z **ETypedElement** je nastaven na specifickou hodnotu.
required	true |false	Pokud je *true*, **lowerBound** atribut z **ETypedElement** je nastaveno na 1. Jinak je nastaveno na 0.
type	Type-name	**eType** reference z **ETypedElement** je nastaven na **EClassifier** příslušného Java type-name.
unique	true | false	**unique** atribut z **ETypedElement** je nastaven na specifickou hodnotu.
upper or upperBound	Integer-value	**upperBound** atribut z **ETypedElement** je nastaven na Integer-value. Specifikovaná hodnota musí být větší než 0, nebo -1 (unbounded).

Tab. 3: Vlastnosti pro Typed Element

@model Vlastnosti pro Strukturální Vlastnosti

EStructuralFeature je specifikován jako přistupující objekt metody v interface příslušné třídy **EClass**, která obsahuje danou vlastnost. Vlastnosti pro **ETypedElement** jsou také platné.

Vlastnost	Hodnota	Použití
dataType	Data-type	Specifický **EDataType** pojmenovaný data-type se využívá jako **eType** pro **EAttribute**. Pokud není namodelován, **EDataType** je vytvořen s patřičným názvem.
default or defaultValue	Default-value	**defaultValueLiteral** atribut z **EAttribute** je nastaven na string hodnotu identifikovanou default-value.
id	true |false	**id** atribut z **EAttribute** je nastaven na specifickou hodnotu.

Tab. 4: Vlastnosti pro Strukturální Vlastnosti

@model Vlastnosti pro Reference

EReference je specifikováno jako metoda v interface příslušné třídy **EClass**, která obsahuje referenci. Vlastnosti pro **EStructuralFeature** a **ETypedElement** jsou také platné.

Vlastnost	Hodnota	Použití
containment	true |false	**containment** atribut z **EReference** je nastaven na specifickou hodnotu.
opposite	Reference name	**opposite** reference z **EReference** je nastavena na odpovídající **EReference** specifikovanou reference-name. **opposite** musí patřit **EClass**, Která je identifikována **eType** z dané **EReference**.
resolveProxies	true |false	**resolveProxies** atribut z **EReference** je nastaven na specifickou hodnotu. Výchozí hodnota je *false*, když **containment** je *true*, jinak je hodnota *true*.
type	Type-name	**eType** reference z **EReference** je nastavena na **EClass** příslušné k Java type-name.

Tab. 5: Vlastnosti pro Reference

@model Vlastnosti pro Operace

EOperation je specifikován jako metoda v Java interface příslušné **EClass**, která obsahuje operace. Vlastnosti pro **ETypedElement** jsou také platné.

Vlastnost	Hodnota	Použití
dataType	Data-type	Specifický **EDataType** pojmenovaný data-type je použit jako **eType** pro **EOperation**. Pokud není namodelován, **EDataType** je vytvořen s patřičným názvem.
exceptions	List-of-types	list-of-types je seznam názvů oddělený mezerou, každý je specifikovaný **EDataType**, aby byl využit s příslušnou **eException**. Pokud není namodelován, **EDataType** je vytvořen s patřičným názvem. Pro vyhnutí se specifikaci patřičné **EDataType** pro příslušnou výjimku, "-" znak se může objevit jako položka seznamu.
type	Type-name	**eType** reference z **EOperation** je nastavena na **EClassifier** příslušné k Java type-name.

Tab. 6: Vlastnosti pro Operace

2.6 Specifikace UML

UML získalo rozsáhlou podporu a patří mezi nejrozšířenější modelovací nástroje. EMF se zabývá jen malou podmnožinou UML, není potřeba velkých znalostí UML pro definici Ecore Modelu za použití UML. Příslušné pojmy v UML jsou v podstatě ty, které jsou použity v třídním diagramu: package(balíček), třídy, atributy, asociace a operace.

2.6.1 UML Packages

Vrchní element Ecore Modelu je **EPackage** a každý UML package je mapován do **EPackage**.

- Atribut **name** z **EPackage** koresponduje se jménem UML package.
- **eClassifiers** reference je naplněna na základě UML tříd obsažených v UML package. Typ každé **EClassifier** vytvořené na základě příslušné UML sterotype tříd.
- **eSubpackages** a **eSuperPackage** reference jsou nastaveny podle analogie UML vztahů.
- Atributy **nsURI** a **nsPrefix** nemohou být vyjádřeny přímo v UML.

2.6.2 UML Specifikace pro EClassifier

Třídy

Každá UML třída má stereotyp <<interface>> nebo ne, je mapována do **EClass** v **EPackge**, který koresponduje s UML package třídy.

- Atribut **name** z **EClass** je stejný jako jméno příslušné UML třídy.
- Seznam referencí **eSuperTypes** z **EClass** přísluší UML třídám, které jsou cílem vztahu generalizace.
- **eStructuralFeatures** reference jsou naplněny **EAttribute** a **EReference** odvozených z UML atributů a asociací.
- **eOperations** reference je naplněna třídami **EOperation** odvozených z UML operací třídy.
- Atribut **instanceClassName** je nastaven na jméno UML atributu, který má stereotyp <<javaclass>>.
- Atribut **abstract** je nastaven na *true*, pokud UML třída je abstraktní.
- Atribut **interface** z třídy **EClass** je nastaven na *true*, pokud UML třída má stereotype <<interface>>

Enumerated Types

UML třída se stereotypem <<enumeration>> je mapována do **EEnum**.

- Atribut **name** z třídy **EEnum** je stejný jako jméno příslušné UML třídy.
- **eLiterals** reference je naplněna **EEnumLiterals** odvozených z atributů, které patří UML třídě. Jméno každé třídy **EEnumLiteral** je nastaveno jako jméno UML atributu.

Data Types

UML tříd se stereotypem <<datatype>> je mapován do **EDataType**.

- Atribut **name** z **EDataType** je stejný jako jméno příslušné UML třídy.

- Atribut **instanceClassName** je nastaven na jméno UML atributu třídy, který musí mít stereotype <<javaclass>>.
- **serializable** atribut je nastaven na *false*, pokud je UML třída abstraktní.

2.6.3 UML Specifikace pro Atributy

Každý atribut UML třídy je mapován do **EAttribute** příslušné **EClass**.

- Atribut **name** z **EAttribute** je stejný jako jméno příslušného UML atributu.
- **eType** koresponduje s typem UML atributu.
- Výchozí nastavení atributů **lowerBound** je 0 a **upperBound** je 1.
- Atribut **defaultValueLiteral** je nastaven podle inicializační hodnoty UML atributu, pokud je specifikována.
- Atribut **derived** je nastaven na *true***,** pokud UML atribut je označen jako odvozený.
- Výchozí nastavení atributů **transient, volatile, unsettable,** a **iD** je *false*. Pro atributy **unique** a **changeable** je výchozí nastavení *true.*

2.6.4 UML Specifikace pro Reference

Každý směrovatelný konec UML asociace mezi dvěma třídami je mapován na **EReference** příslušné **EClass**.

- Atribut **name** z **EReference** je stejný jako jméno role příslušného konce UML asociace.
- **eType** je **EClass** příslušná k UML třídě na druhém konci asociace.
- Atributy **lowerBound** a **upperBound** jsou odvozené z mnohočetnosti příslušné UML asociace.
- Pokud UML asociace je obousměrná, reference **eOpposite** příslušných dvou **EReference** odkazuje jedna na druhou.
- Uzavřená asociace je nastavena na *true*, pokud příslušný konec UML asociace je složený nebo souhrnný.
- Atribut **derived** je nastaven na *true*, pokud UML asociace je označena jako odvozená.
- Výchozí nastavení atributů **transient, volatile, unsettable** je *false*. Pro atributy **resolveProxies** a **changeable** je výchozí nastavení *true.*

2.6.5 UML Specifikace pro Operace

Každá operace v UML třídě je mapována do **EOperation** příslušné **EClass**.

- Atribut **name** z **EOperation** je stejný jako jméno UML operace.
- Reference **eType** je **EClassifier** příslušný k návratovému typu UML operace.
- Výchozí nastavení **lowerBound** je 0 a **upperBound** je 1.
- **eParameters** reference je naplněna **EParameter** odvozených z příslušných parametrů UML operace.
- **eExceptions** reference zahrnuje **EClassifier** příslušný ke specifikaci výjimky UML operace.

- Atribut **unique** je nastaven na výchozí hodnotu *true.*

2.6.6 UML modely implementace

Z definice EMF platformy vyplývá možnost vytvářet Ecore model, tudíž základ pro implementaci notací na platformu EMF, z UML modelu. Výše popsané UML specifikace je možné převést do Ecore modelu, ale také i zpětně z Ecore modelu. V rámci implementace je Ecore model využit pro definování metamodelů notací a v první fázi implementace jsou tyto metamodely definovány přímo v Ecore. Pro odzkoušení možností EMF platformy, byly vygenerovány UML modely dodatečné z Ecore metamodelů notací. Ve zdrojových kódech každé notace jsou tyto UML modely uloženy ve složce „model“. Níže si předvedeme ukázku UML modelu jak je vyspecifikována v rámci platformy EMF a popsána v předešlých kapitolách specifikací UML.

Ukázka generovaného UML modelu v EMF notace BPMN

Obr. 19: BPMN UML model na platformě EMF

2.7 Využitelnost Ecore Modelu

1. Ecore Model a jeho serializace XMI je základem EMF
2. Ecore Model může být vytvořen ze tří různých zdrojů: UML model, XML Schéma nebo komentované Java interface
3. Implementace Java kódu a volitelně jiné formy modelů mohou být generovány z Ecore Modelu.

2.7.1 Vytváření a Editování Modelu

Pro EMF máme tři reprezentace, z nichž můžeme definovat Ecore Model. Pokud začneme s Java interface nebo s XML Schématem, EMF prozkoumá specifikace a poté z nich vytvoří Ecore Model. Pokud začneme s UML jsou tři možnosti definování modelu:

1. Přímá editace Ecore. EMF obsahuje jednoduchý stromový editor pro Ecore, také existuje v rozšíření Ecore Tools grafický editor založený na UML notaci. Je možné také využít software třetích stran jako je Topcased's Ecore Editor, Omondo's EclipseUML nebo Soyatec's eUML.
2. Import z UML. EMF Project a EMF Model průvodce (wizard) poskytuje rozšiřující Framework, do kterého mohou být importovány modely. EMF poskytuje podporu jen pro Rational Rose (.mdl).
3. Export z UML. Obdobně jako u druhého bodu, ale konverze je podporována výhradně nástrojem UML. Export je volán výhradně z UML nástroje, namísto EMF průvodce (wizard).

První možnost se zdá nevhodnější, jelikož importní a exportní kroky se nevyskytují ve vývojovém procesu. Jednoduše se nejčastěji využívá editace modelu a poté z něj vygenerování kódu. Další dvě možnosti také vyžadují explicitní reimport a reexport při změně UML modelu. Naproti tomu výhodou importu a exportu je možnost využít více funkcionalit UML nástroje, které poskytuje.

2.7.2 Generování Kódu

Hlavní a nejdůležitější výhoda EMF je urychlení produktivity, která vychází z automatického generování kódu. Dá se říci, že pokud nadefinujeme model, jsme schopni ho během chvíle vygenerovat. Zjednodušeně řečeno stačí si vytvořit EMF Project za pomoci průvodce, který automaticky spouští generátor.

1. Generování tříd modelu. V EMF Ecore třída odpovídá dvěma věcem v Javě, může to být interface nebo odpovídající implementační třída.
2. Oblasti obnovy a slučování. EMF generátor produkuje soubory, které jsou kombinací generovaného a ručně psaného kódu.
3. Generátor Modelu. Většina dat potřebných pro EMF generátor jsou uloženy v Ecore Modelu, například generované třídy a jejich jména atributy a reference a další.
4. Generátor adapterů a Plug-inů

2.8 Modelovací Standardy EMF

EMF (Eclipse Model Framework) je spojován s několika důležitými modelovacími standardy definovaných organizací Object Management Group (OMG). Mezi tyto standardy patří UML, MOF, XMI a MDA. Nyní si popíšeme tyto standardy a také vzájemný vztah k EMF viz kapitola 3.

3 Metamodelování

Metamodelování je definováno jako modelování meta-modelů. Metamodelem rozumíme popis struktury a vlastností daného modelu, neboli také schéma modelu. Výhodou metamodelování je konzistentní způsob popisu odlišných datových struktur, z nichž se tvoří jednotný vyšší interoperabilní systém. V praxi metamodel vychází z popisu modelu jako systému, používaného pro vývoj aplikací, informačních systému a simulací výše zmíněných business procesů. Metamodelování vychází z principů čtyřvrstvé architektury.

1. Vrstva dat reprezentuje fyzicky popisovaná data. Konkrétní instance objektů, mohou to být záznamy z databáze XML soubory.
2. Vrstva modelu obsahuje metadata, jež jsou shrnuta do modelů a určují strukturu dat. Může se jednat o UML diagram tříd nebo jiný konkrétní model modelovacího jazyka.
3. Vrstva metamodelu zahrnuje popis definující strukturu a sémantiku metadat. V této úrovni se nacházejí metamodely specifických modelovacích jazyků.
4. Vrstva meta-metamodelu určuje sémantiku a strukturu meta-metadat, čímž popisuje metamodel.

Metamodelování dovoluje popsat jednotným způsobem odlišné datové struktury systémů, jež umožňují struktury skládat dohromady ve vyšší systém, který dokáže pracovat s daty systémů, ze kterých je sestaven pro jejich integraci. Metamodel také napomáhá zvládat komplexní projekty, pro lepší porozumění problematiky rozsáhlých systémů. [1]

3.1 Unified Modeling Language

UML je nejpopulárnější standard pro popis systémů, jež jsou založeny na konceptu. Využívá se ke specifikaci a návrhu softwaru, zejména softwaru psaného v objektově-orientovaném jazyce. Hlavní myšlenkou je v UML, že komplexní systémy je nejlepší popisovat pomocí několika různých pohledů, jelikož jediný pohled nemůže zachytit všechny aspekty systému. UML zahrnuje několik různých typů modelů diagramů k zachycení scénářů, struktury tříd, chování a implementace. [3], [5]

EMF se zabývá jen jedním typem diagramu UML. Tímto typem je třídní diagram. Toto zaměření není v žádném případě zavržení celostního přístupu UML. Lépe řečeno je to výchozí bod pro realizaci systému, jež díky UML diagramům může být vyjádřen velmi komplexně a implementace by byla příliš složitá.

3.1.1 Třídní diagram

Nejběžnějšími diagramy UML, se kterými se setkáváme, jsou třídní diagramy. Třídní diagram je sestaven z tříd a vztahů. Popisuje strukturu systému. Třídy mohou reprezentovat informace, produkty, dokumenty nebo organizace. Třídy diagramu jsou propojeny pomocí asociací různého typu, mohou to být asociace, agregace, kompozice, dědičnost, závislost a realizace. Balíčky (packages) jsou používány k organizování a zvládnutí komplexností rozsáhlých modelů. Balíček může sdružovat třídy, stavy a aktivity, které mohou být odkazovány jako celek.

3.2 Meta-Object Facility

Meta-Object Facility (MOF) konktrétně definuje podmnožinu UML pro popis třídy modelující koncept do objektu repositáře. MOF je srovnatelné z Ecore Modelem. Se zaměřením na integrační nástroj, spíše než na řízení repositáře metadat, Ecore se vyhýbá složitostem MOF, což je použitelné pro optimalizaci implementace Ecore Modelu. Ecore Model a MOF má mnoho společného ve schopnosti specifikovat třídy a jejich strukturální vlastnosti, vlastnosti chování, dědičnost a zrcadlení. Odlišují se v oblasti životního cyklu, strukturách datových typů, vztahů mezi balíčky (package) a komplexnímu aspektu asociací. Essential Meta-Object Facility (EMOF) je odlehčené jádro metamodelu, který je docela podobné Ecore Modelu. Vzhledem k tomu jak jsou si oba modely podobné, EMF je schopné podporovat EMOF přímo jako náhradu XMI serializace Ecore Modelu. [5]

3.3 XML Metadata Interface

XMI je standard, který spojuje modelování s XML, definuje jednoduchou cestu k serializaci modelu do XML dokumentu. Hlavním úkolem si tedy klade definovat způsob, jak reprezentovat modely ve formě XML dokumentů a umožnit jejich přenositelnost. XMI vytváří vazbu mezi metamodelem a schématem XML a zároveň mezi modelem a reprezentovaným XMI. Vazba je popsána pravidly, která definují, jaké elementy XML odpovídají jednotlivým částem modelu. Díky tomuto mechanismu je možno automaticky generovat schémata XML dokumentů pro libovolný metamodel. XMI může být také využit pro výměnu informací o datových skladech. Je to nadstavba nad normami Meta Object Facility (OMG), Extensible Markup Language (W3C) a Unified Modeling Language (OMG) (srov. XML, UML). Struktura XMI dokumentu se shoduje s daným modelem, stejnými názvy a hierarchií prvků, který odpovídá uzavřené hierarchii modelu. V důsledku toho, je vztah mezi modelem a jeho XMI serializací snadno pochopitelný. Přestože XMI může být, a je ve výchozím nastavení používáno jako formát pro serializaci instance jakékoliv EMF modelu, je velmi vhodný pro použití s modely představující metadata, tj. metamodely, jako je Ecore Model. Obecně účelem XMI je umožnit snadnou výměnu metadat mezi UML-modelovacími nástroji a repositáři metadat založených na MOF v distribuovaných heterogenních prostředích. Integrace tří norem XML, UML a MOF do XMI umožňuje vývojářům systémů sdílení objektů modelů a metadat. [5]

3.4 Model Driven Architecture

MDA poskytuje postupy a umožňuje vznik nástrojů pro specifikace systémů nezávisle na platformě, která je podporuje, specifikace platforem, výběr konkrétní platformy pro konkrétní systém a transformace specifikace systému do podoby uzpůsobené konkrétní platformě. Pod pojmem model MDA chápeme formální specifikaci systému ve standardu UML. Z pohledu jeho strojového zpracování se pak jedná o jeho reprezentaci v jazyku eXtensible Model Interchange (XMI), případně EMF Ecore.

EMF podporuje MDA koncept využití modelů jako vstup pro vývoj a integrační nástroje v EMF. Model se používá k řízení generování kódu a serializaci pro výměnu dat. [5]

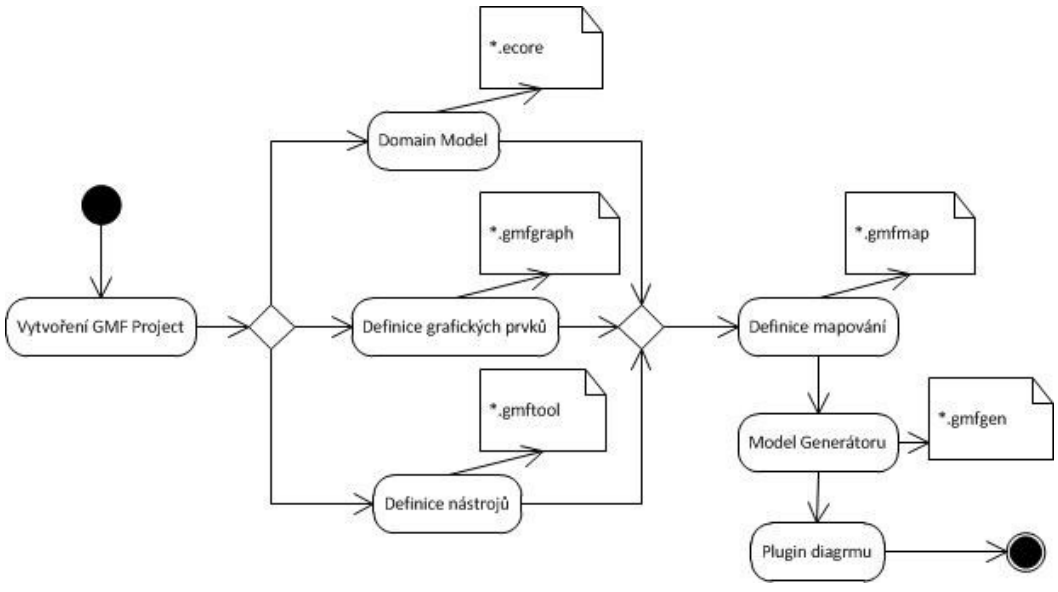
4 Praktická část

4.1 Úvod k praktické části

Cílem praktické části je vývoj modelu BPStudia a dalších notací byznys modelování pro implementaci na platformě EMF. Pro grafické zobrazení je v rámci implementace využita platforma GMF (Graphical Modeling Framework), která umožňuje propojení metamodelu notací s grafickou reprezentací a umožňuje vytvořit nástroje pro modelování byznys procesů jednotlivých notací. Tyto nástroje jsou součástí pluginů, které při instalaci do vývojového rozhrání Eclipse, zpřístupní vytváření diagramů a modelů. Dále tyto pluginy obsahují funkce pro transformace procesů mezi jednotlivými notacemi a také import / export nástroj, který načítá XML soubory jednotlivých notací. Vývoj transformačních, importních a exportních nástrojů je součástí práce kolegy Bc. Ondřeje Führera a v mé praktické části toto nebude popisováno. Součástí mé práce je integrace těchto nástrojů do pluginů a vytvoření transformačního mezi-metamodelu, přes který se realizují všechny transformace.

V praktické části se zaměříme na definici prerekvizit modelovacích a implementačních nástrojů, vývoj na platformě EMF a GMF, definici metamodelů, jak jednotlivých notací, tak i transformačního mezi-metamodelu.

Z praktického hlediska pokud se zaměříme na implementaci konkrétní notace je potřeba definovat nejdříve metamodel na platformě EMF. Tento metamodel je definován jako Ecore Model. V rámci práce s EMF a Ecore Modelem je možné generovat implementační třídy, které nám umožní pracovat v UI Eclipse s konkrétními operacemi a funkcemi modelu. Dále je využita platforma GMF, která k danému modelu Ecore definuje grafické prvky pro vytváření jednotlivých instancí notace a umožní grafické modelování byznys procesů. Aktivitní diagram (Obr. 20) zobrazuje jednotlivé aktivity potřebné k vývoji grafického nástroje na platformě GMF. Jednotlivé aktivity jsou blíže rozepsány v kapitole 4.3.2.



Obr. 20: Vývoj na platformě GMF

4.2 Prerekvizity implementace

Pro implementaci praktické části je využito vývojového nástroje Eclipse. Standartní verze Eclipse neobsahuje komponenty pro vývoj na platformě EMF nebo GMF a je potřeba instalace těchto komponent. Nejenom pro vývoj, ale i pro následné využití pluginů vytvořených v rámci této diplomové práce je potřeba doinstalovat patřičné komponenty. Instalace a uživatelská příručka je součástí přílohy, v této kapitole si jen stručně popíšeme verze a nástroje, které jsou využity pro implementaci.

4.2.1 Eclipse

Pro instalaci Eclipse je výběr ze dvou možností:

- Eclipse Classic 3.7.2
- Eclipse Modeling Tools - Version: Indigo Service Release 2

Instalační zdroje je možné stáhnout z http://www.eclipse.org/downloads/.

4.2.2 Instalované komponenty

Po instalaci IDE Eclipse je potřeba také doinstalovat patřičné komponent. V Eclipse zadejte *Help > Install New Software*.

URL adresa pro stahování komponent: Indigo - http://download.eclipse.org/releases/indigo.

Pro Eclipse Classic:

- Eclipse Modeling Framework (EMF)
- Graphical Modeling Framework (GMF)

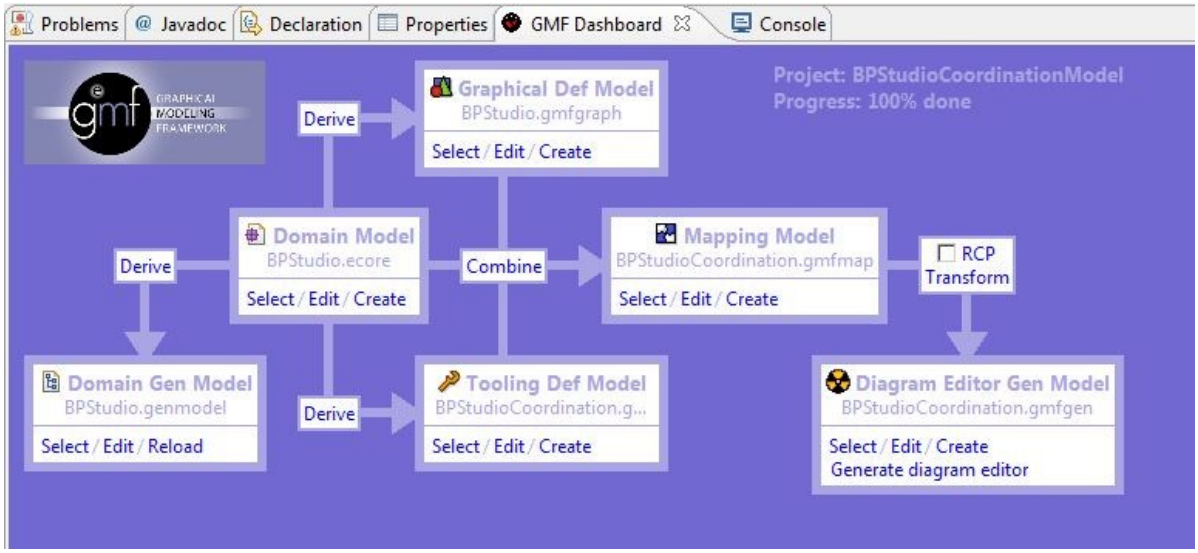
Pro Eclipse Modeling Tools

- Graphical Modeling Framework (GMF)

4.3 Vývoj na platformě EMF a GMF

Všechny modelovací nástroje notací byznys procesů jsou vytvořeny pomocí velice flexibilních a komplexních nástrojů Eclipse Modelling Framework a Graphical Modelling Framewok. EMF je framework pro modelování metamodelů. GMF vychází z EMF a pracuje s metamodely v něm vytvořených. Vývoj projektu v rámci těchto frameworků je postaven tak, že pomocí EMF je definován metamodel jako model Ecore, ten je pak převzat GMF, kde jsou konfigurovány modely GMF, které zajišťují požadované zobrazení jednotlivých diagramů. Pro definování chování diagramů je také někdy nutné zasáhnout do generovaného kódu, který je velice obsáhlý. [9], [10]

GMF Dashboard, který je dostupný v rámci GMF projektu vede vývoj projektu a jeho součástí. Přes tento nástroj je možné také generovat jednotlivé modely GMF, které je potřeba posléze zpracovat do požadované formy. Automaticky generované modely GMF slouží jen jako šablona pro následující práci s modely.



Obr. 21: GMF Dashboard

4.3.1 EMF

Eclipse Modelling Framework je nástroj modelování, který umožňuje jednoduché generování kódu, který umožňuje vývoj aplikací založených na strukturovaném datovém modelu. EMF generuje implementační třídy v jazyce Java a poskytuje nástroje pro editování jejich modelů. Metamodely mohou být v EMF navrženy několika způsoby UML, XML, Java interface, Ecore Model. Za pomoci jedné z těchto reprezentací je možné generovat reprezentace ostatní a k tomuto také patřičné implementační třídy. EMF sjednocuje technologie XML, UML a Java díky EMF Ecore Modelu. Všechny tyto možnosti umí EMF zpracovat a vytvořit z nich tentýž metamodel. K unifikované reprezentaci metamodelů se používá metamodel zvaný Ecore Model, který je svým vlastním metamodelem. Za použití Ecore Modelu jsou navrženy všechny metamodely notací v praktické části. Také pro další vývoj na platformě GMF je potřeba nejprve navrhnout jednotlivé metamodely (Domain Model) a z nich vygenerovat v rámci EMF.Edit implementační třídy editoru modelu (Domain Gen Model). [11], [7]

Domain Model (Metamodel)

Metamodel Ecore je návrhový model vycházející ze základních objektů **EObject**. Jsou zde definovat třídy, jejich atributy nejrůznějších datových typů a asociace včetně násobností. Vlastní třídy jsou definovány jako **EClass.** Uvnitř těchto tříd jsou atributy **EAtribute** libovolného typu **EDataType**. Mezi třídami je možné definovat vztahy, které jsou reprezentovány třídou **EReference**. Třída **EReference** reprezentuje vždy jeden konec asociace mezi třídami, omezením pro návrh asociací je, že je možné navrhnout asociace jen v jednom směru. Obousměrné reference se modelují za použití dvou referencí. V rámci EMF je možné definovat a editovat Ecore model pomocí několika

nástrojů. Metamodel je možné definovat ve stromové struktuře nebo za použití modelovacího nástroje diagramu tříd.

Domain Gen Model

Z metamodelu Ecore je možné generovat EMF Generator Model. Tyto dva modely jsou poté základem pro automaticky generovaný Java kód, který je vytvořen na základě jádra EMF.Edit. vygenerovaný Java kód slouží jako základ pro práci s modelem notace a také v rámci GMF projektu s diagramem, který modeluje danou notaci graficky. Jádro EMF.Edit generuje tyto části implementačních tříd:

- Model Code
- Edit Code
- Editor Code

Model Code umožňuje práci a zobrazení modelu ve stromové struktuře. **Edit Code** vytváří implementační třídy jednotlivých prvků pro editaci modelu a **Editor Code** definuje operace a funkce nad modelem. **Editor Code** je využit pro integraci transformačních a importních nástrojů.

4.3.2 GMF

Graphical Modeling Framework je framework, který spojuje nástroj pro práci s EMF v prostředí Eclipse a je jeho grafickou nástavbou. Základní funkcí GMF je pomocí konfiguračních nástrojů, které jsou definovány jako modely, vytvořit grafický editor diagramů na základě metamodelu Ecore. Celý projekt se skládá z různých modelů, které jsou na sobě závislé. Každý model představuje chování nějaké části projektu. Jednotlivé části projektu jsou rozděleny pro návrh, grafiku i samotné chování a každý model pracuje na své části. [8], [9] ,[10]

Graphical Definition Model

Grafický model definuje vzhled každého objektu včetně vzhledu a chování. Soubor gmfgragh obsahuje stromovou strukturu, jejímž kořenem je objekt reprezentující celý diagram. Do něj je možné přidávat prvky, které diagram zobrazuje. Eclipse nabízí množství předdefinovaných vzhledů, z jejichž lze vytvořit různé tvary a definovat jejich vlastnosti a popisné informace.

Tooling Definition Model

Tento model slouží k navržení prostředí modelování. Obsahem souboru gmftool jsou specifikace nástrojů, elementů a menu, které se v diagramu objeví v paletě nástrojů. Je zde možné definovat různé ikony elementů a také je různě seskupovat do menu.

Mapping Model

Model spojující **Domain Model, Graphical Definition Model** a **Tooling Definition Model**. Zde uživatel specifikuje, jak budou objekty metamodelu zobrazeny, resp. na jaký grafický prvek budou vázány.

Diagram Editor Gen Model

Posledním krokem je vytvoření modelu gmfgen, který vše předešlé spojuje do jednoho modelu. Z tohoto modelu se automaticky vygeneruje kód zásuvného modulu pro diagram. V tomto kódu je možné provádět úpravy, zejména úpravy zobrazení prvků, které předešlé nástroje neumožňují.

4.4 GMF architektura řešení

Jak je zmíněno v předešlé kapitole, GMF definuje několik specifických modelů. Některé z těchto modelů jsou inicializovány při spuštění pluginů a jsou využívány generovaným diagram editorem. Mezi tyto modely patří Notation model, Graphical Definition model, Tooling Definition model, Mapping model a Generator model. [8], [9] ,[10]

4.4.1 Notation Model

Notation model je využit pro uložení vizuálních informací výkresu diagramu a je nezávislý na základním sémantickém modelu. Například se využívá pro uložení informací o pozici a velikosti elementů či barevných stylech. Tento model je založen na těchto třídách: Diagram, Node (uzel), Edge (hrana). Každá z těchto tříd slouží k uložení příslušných vlastností pro specifický diagram, node a edge v jejich vizuální reprezentaci. Notation model může být také rozšířen o specifické data definované uživatelem. V našem případě implementace třídu Diagram reprezentuje konkrétní notace (EPC, BPMN, Diagram aktivit, BP Studio modely), Node reprezentuje elementy notací (aktivity, události, stavy) a Edge reprezentuje hrany neboli přechody. V implementaci metamodelů je přihlédnuto k těmto třídám a jednotlivé metamodely jsou navrženy tak, aby korespondovaly jednotlivé elementy s těmito třídami.

Každý otevřený editor diagramu má připojenou svou instanci notation modelu a je aktivně využíván při běhu editoru jako API pro přenos specifických informací diagramu a ukládá příslušné informace spuštěné instanci editoru za použití standartních perzistentních mechanismů EMF. [8]

4.4.2 Graphical Definition Model

Graphical Definition Model je instancí pro shromažďování informací o implementovaném grafickém editoru diagramu, o jeho grafických objektech a plní příslušnými parametry generátor kódu subsystému. Tento model není přístupný v běžícím prostředí editoru diagramu, slouží jen pro generování kódu při vytváření editoru diagramu. GMF obsahuje několik takovýchto modelu popisujících budoucí editor diagramu. Tento model popisuje vlastnosti a vytváří jednotlivé elementy diagramu, kterými jsou Nodes (aktivity, události, stavy), Connections (přechody, hrany, spojení elementů), případně Compartments (oddělení) a Labels (popisky). Elementy diagramu mohou být vytvořeny ze standardů GEF (Graphical Editing Framework poskytuje bohatý nástroj pro vytváření grafických prvků a pohledů na platformě Eclipse) nebo jsou vytvořeny pomocí takzvaných „Custom figures“, které umožňují vlastní grafické nastavení elementů za pomoci ručně implementovaného rozhraní IFigure. Obě tyto možnosti umožnují využít k uspořádání vnořených údajů vlastní rozvržení dat elementu (Custom Layout Data). Informace uložené v tomto modelu jsou dostačující ke generování kódu popisující složené vlastnosti. Generované vlastnosti mohou být inicializovány

editorem diagramu a využity pro vizualizaci elementů diagramu. Tento model plně popisuje zobrazení elementů v diagramu, ale nepopisuje aspekty chování generovaných elementů. Tyto aspekty chování mohou být ručně dopsány do generovaného kódu. V rámci implementace jsou použity zejména vlastní třídy elementů, které popisují charakter daného elementu. [8]

4.4.3 Tooling Definition Model

Dalším důležitým aspektem pro popis editoru diagramu je definice palety diagramu (toolbar). Definuje jednotlivé prvky, jako jsou tlačítka, menu a UI akce. Tento aspekt diagramu obsahuje Tooling Definition model. V základu model poskytuje definici nástrojů a akcí editoru diagramu, kterým můžeme diagram ovládat. Například tlačítka pro vložení nového elementu do diagramu nebo také základní funkční tlačítka pro zoom, výběr elementu. Model umožňuje také vytvářet skupiny, do kterých tyto tlačítka můžeme sdružovat, či definovat vlastnosti ikon. Na základě informací uložených v tomto modelu je generován takzvaný PaletteFactory kód. Tento model je odkazován do Mapping modelu pro generování obstojného kódu, který již daným tlačítkům a akcím přiřazuje patřičné chování. V implementaci tento model definuje tlačítka elementů jak pro aktivity, události a stavy, tak pro jednotlivé přechody, tudíž všechny elementy notací EPC, BPMN, Diagramu aktivit a BP Studia. [8]

4.4.4 Mapping Model

GMF rozlišuje mezi dvěma nezávislými aspekty definice diagramu – Graphical a Tooling. Dalším důležitým aspektem digramu notace, který vytváříme je sémantický metamodel. GMF je navrženo k vytváření diagramů na základě EMF, tedy zahrnuje definovaný EMF model (.ecore). K doplnění popisu diagramu je nezbytné tyto tři aspekty spojit dohromady a toto má za úkol Mapping model. Při práci s definicí mapování tohoto modelu vytváříme mapovací elementy, které jsou CanvasMapping, NodeMapping, LinkMapping. Toto mapování se používá k úplnému popisu plochy diagramu. Každý mapovací element je propojen do elementu Ecore modelu tedy do Domain modelu, společně s příslušnou vizuální reprezentací z Graphical Definition modelu a potřebnéhonastavení nástrojů, které tento element ovládá z Tooling modelu. LinkMapping a NodeMapping také obsahují LabelMapping, které se využívají k zobrazení atributů v sémantickém modelu elementů jako popisky diagramu. Přesněji řečeno v implementaci notací je LabelMapping využit pro zobrazení názvů a popisků například aktivit, událostí nebo jejich přechodů. LabelMapping udržuje referenci atributu ze sémantického modelu do elementu v Graphical Definition modelu. NodeMapping může mít mapovací element CompartmentMapping, který se používá k logickému seskupení elementů. Také CompartmentMapping udržuje referenci atributu ze sémantického modelu do elementu v Graphical Definition modelu. NodeMapping a LinkMapping nesou potřebné informace prezentující příslušné elementy, jako jsou aktivity, události, přechody a jejich pojmenování na uzly, linky a popisky v diagramu. Další mapovacím elementem je ChildReference, který zachycuje informace o vazbě rodič-potomek mezi různými uzly diagramu. [8]

Každý Mapping model reprezentuje popis diagramu, zatím co Tooling a Graphical modely jsou použity k popisu části aspektů celé definice diagramu. Stejné Tooling a Graphical modely mohou být využity pro různě definované Mapping modely a tudíž výsledek mohou být i různé diagramy.

4.4.5 Generator Model

Graphical, Tooling Definition a Mapping model společně s EMF metamodelem (.ecore) popisují kompletně konečnou strukturu diagramu. Tyto modely organizují specifický jazyk pro definici diagramu. Tento jazyk může být úspěšně použit ke shromáždění potřebných informací pro jednotlivé nástroje modelů, ale ne pro dobře generovaného kódu. Generátor kódu potřebuje více specifických detailů pro generování, jako jsou jména tříd a balíčku pro příslušné generované třídy a také implementační specifikační parametry umožňující jemné ladění generovaného kódu. Generator model pracuje s elementy, jako jsou GenPlugin (využívaný k parametrům generování Eclipse pluginu), GenDiagram (využívaný ke generování editoru diagramu) a GenClass (využívaný ke generování GEF elementů EditParts a EditPolicies). Účel tohoto modelu je odrážet potřebné detaily a specifika procesu generování kódu, stejně tak jako proces transformace GMF modelů na Generator model. Generator model je srovnatelný s EMF .genmodelem, jelikož oba popisují stejný koncept. Stejně tak jako u EMF, může být GMF Generator model automaticky vytvořen z modelů vyšší úrovně. Samozřejmě transformace z modelů vyšší úrovně GMF na Generator model není transformace jedna ku jedné jako u EMF. Tato transformace je prováděna naprogramovanými Java třídami, tudíž je možný určitý stupeň úprav. Většina kódu diagramu může být generováno bez modifikací na Generator modelu, který automaticky vytvoří transformační proces na výsledný diagram. Za běžné menší úpravy, které byly v rámci implementace prováděny, jsou změna pojmenování, popisu, verze pluginů a dalších popisných atributů diagramu. [8]

4.5 Metamodely

V rámci vývoje je potřeba definovat jednotlivé metamodely notací pro byznys procesy. Tyto metamodely jsou navrhnuty pro potřeby implementace do EMF jako Ecore metamodely. Také jsou zohledněny požadavky pro zavedení těchto metamodelů do projektu GMF. Z hlediska GMF je potřeba modelovat třídy pro jednotlivé prvky a asociace v rámci notací. Prvky použité pro modelování metamodelů: [1]

- Třídy – reprezentují jednotlivé entity metamodelu.
- Agregace – reprezentuje prvky, které obsahuje diagram notace.
- Asociace – v rámci metamodelu určují vlastnosti vazeb mezi prvky notací.
- Generalizace – pro lepší a přehlednější modelování metamodelů.

4.5.1 EPC

EPC je určen zejména pro zobrazení průběhu procesu na úrovni činností větvení cest k návaznosti na další procesy a přiřazení odpovědností za výkon jednotlivých činností. Jedná se o velmi rozšířenou metodu, zejména z důvodu nasazení a podpory velkými softwarovými systémy jako jsou ARIS. [12]

Základní entity EPC:

Activity – aktivity definují, co má být v danou chvíli v rámci procesu provedeno.

Event – událost popisuje situaci před nebo po aktivitě.

Logical Operation – logické spojky spojují události a aktivity, jedná se o AND, OR a XOR.

Třídy reprezentující EPC entity:

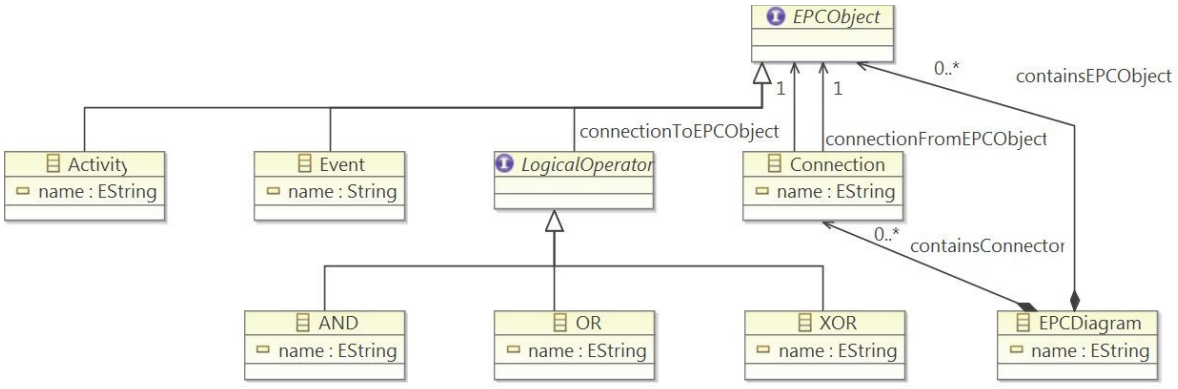
- EPCDiagram – třída reprezentující diagram
- EPCObject – abstraktní třída
- Connection – obsahuje atribut **name:EString**
- LogicalOperation – abstraktní třída
- AND – obsahuje atribut **name:EString**
- OR – obsahuje atribut **name:EString**
- XOR – obsahuje atribut **name:EString**
- Event – obsahuje atribut **name:EString**
- Activity – obsahuje atribut **name:EString**

Agregace mezi entitami:

- containsEPCObject
- containsConnector

Asociace mezi entitami:

- connectionToEPCObject
- connectionFromEPCObject



Obr. 22: Metamodel EPC

4.5.2 Diagram Aktivit

Diagram aktivit popisuje dynamiku vyvíjeného systému a podává zjednodušený přehled jednotlivých kroků nějaké operace nebo procesu. Diagram se skládá z následujících elementů: počáteční a koncový stav, aktivita, přechod, alternativní větvení a spojení, paralelní větvení a spojení a rozhodovací blok. Tyto základní elementy jsou převedeny do metamodelu diagramu jako třídy, které poté GMF využívá jako grafické prvky. [3]

Třídy reprezentující entity Diagramu Aktivit:

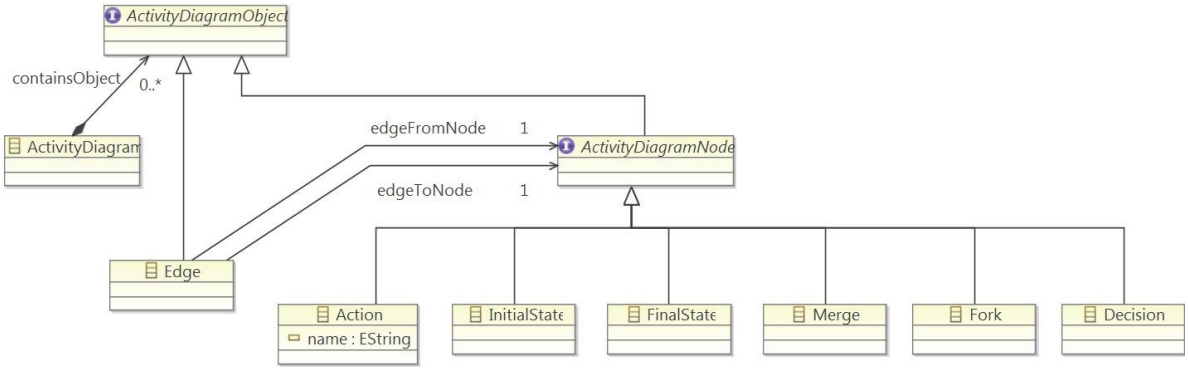
- ActivityDiagram – třída reprezentující diagram
- ActivityDiagramObject – abstraktní třída
- Edge
- ActivityDiagramNode – abstraktní třída
- Action – obsahuje atribut **name:EString**
- InitialState
- FinalStage
- Merge
- Fork
- Decision

Agregace mezi entitami:

- containsObject

Asociace mezi entitami:

- edgeFromNode
- edgeToNode



Obr. 23: Metamodel Diagramu Aktivit

4.5.3 BPMN

V rámci BPMN využíváme diagram zvaný Business Process Diagram (BPD). [2] BPMN je modelovacím nástrojem vhodným pro modelování byznys procesů hlavně pro svou přehlednost. Model obsahuje grafické objekty, kterým jsou zejména aktivity a toky informací mezi nimi. Grafická zobrazení jednotlivých prvků má přesně definované tvary, které může odlišovat v rámci modelování barvami. Pro naši implementaci notace BPMN na platformu EMF jsou barvivy elementů předdefinovány.

Business Process Diagram obsahuje čtyři základní druhy grafických elementů, ty se poté dělí na podtypy. V části níže jsou popsány všechny prvky, které jsou využity pro definovaný metamodel BPMN. Z metamodelu a jeho entit jsou patrné všechny elementy, které jsou implementovány.

Základní entity BPMN:

Flow Object – objekt související s tokem informací v procesu.

Event – událost, která označuje začátek, konec nebo událost v průběhu procesu.

Activity – definuje činnost v procesu.

Gateway – označuje větvení či souběh toků procesu.

Connection – slouží ke spojení objektů či artefaktů.

- Sequence Flow – určuje pořadí aktivit.
- Message Flow – definuje tok zpráv v procesu.
- Association – spojuje objekt s informací.

Artifact – nese upřesňující informace pro proces.

- Data Object – určuje data aktivit.

Swimlane – slouží k uspořádání procesu podle určitých kritérií.

Pool – účastník v procesu.

Lane – podmnožina Poolu, slouží ke kategorizaci.

Třídy reprezentující entity BPMN:

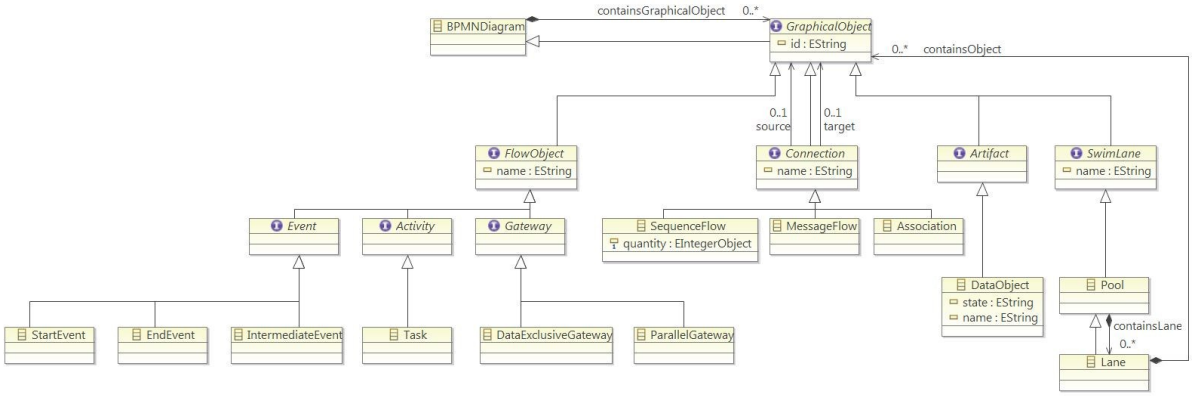
- BPMNDiagram – třída reprezentující diagram
- GraphicalObject – abstraktní třída, obsahuje atribut **id:EString**
- FlowObject – abstraktní třída, obsahuje atribut **name:EString**
- Event – abstraktní třída
- Activity – abstraktní třída
- Gateway – abstraktní třída
- StartEvent
- EndEvent
- IntermediateEvent
- Task
- DataExclusiveGateway
- ParallelGateway
- Connection – abstraktní třída, obsahuje atribut **name:EString**
- SequenceFlow
- MessageFlow
- Association
- Artifact – abstraktní třída
- DataObject – obsahuje atribut **name:EString, state:EString**
- SwimLane – abstraktní třída, obsahuje atribut **name:EString**
- Pool
- Lane

Agregace mezi entitami:

- containsGraphicalObject
- containsObject
- containsLane

Asociace mezi entitami:

- source
- target



Obr. 24: Metamodel BPMN

4.5.4 BP Studio

BP Studio (Buissines Process Studio) je aplikace pro modelování procesů pomocí metody BPMN. BP Studio obsahuje několik základních modelů, jedná se o koordinační, objektový a funkční model. V rámci implementace této metody byl vytvořen metamodel, který zahrnuje elementy všech těchto modelů do jednoho metamodelu. Jednotlivé pluginy přislušných notací modelů BP Studia, jsou založeny na stejném metamodelu, který byl využit pro základní Ecore metamodel implementace těchto pluginů. Níže jsou popsány jednolivé elementy a obrázek metamodelu názorně popisuje jednotlivé části modelů BP Studia. [4]

Třídy reprezentující entity BP Studio (Příloha B, Obr. 29):

- BPStudioDiagram – třída reprezentující diagram
- BPModel – abstraktní třída
- Models – abstraktní třída
- Dictionary – abstraktní třída
- Functional – abstraktní třída
- Coordination – abstraktní třída
- Object – abstraktní třída
- EdgeTypeFunctional – abstraktní třída, obsahuje atribut **name:EString**
- Containment

- Collaboration
- Customer
- Product
- Owner
- NodesTypeFunctional – abstraktní třída
- EdgeTypeCoordination – abstraktní třída, obsahuje atribut **name:EString**
- Flow
- ResposibilityRelationship
- InhibitoryRelationship
- NodesTypeCoordination
- EdgeTypeObject – abstraktní třída, obsahuje atribut **name:EString**
- Association
- Aggregation
- Generalization
- Role
- NodeTypeObject – abstraktní třída
- Process – obsahuje atribut **name:EString, specification:EString, external:EBooleanObject**
- Activity – obsahuje atribut **name:EString, specification:EString, scenatio:EFeatureMapEntry**
- Active – obsahuje atribut **name:EString, feature:EString, service:EString**
- Passive – obsahuje atribut **name:EString, feature:EString**

Agregace mezi entitami:

- containsBPModel

Asociace mezi entitami:

- sourceFunctional
- targetFunctional
- sourceCoordination
- targetCoordination
- sourceObject
- targetObject

4.5.5 Transformační mezi-metamodel

Transformační mezi-metamodel byl navržen k transformacím notací. Je to model, který slouží k převedení notace do notace jiné. Model je mezikrokem v transformaci. Každý model je nejdříve převeden do tohoto transformačního modelu a z něj pak do požadovaného modelu určité notifikace. Tento model byl vytvořen na základě požadavků pro jednotlivé transformace kolegy Führera, který implementoval transformační funkce. Toto řešení bylo zvoleno v závislosti na zjednodušení implementace transformací a také pro zlepšení architektury jejího řešení z důvodu snížení počtu jednotlivých transformací. Níže jsou popsány jednotlivé elementy metamodelu.

Třídy reprezentující entity transformační mezi-metamodel:

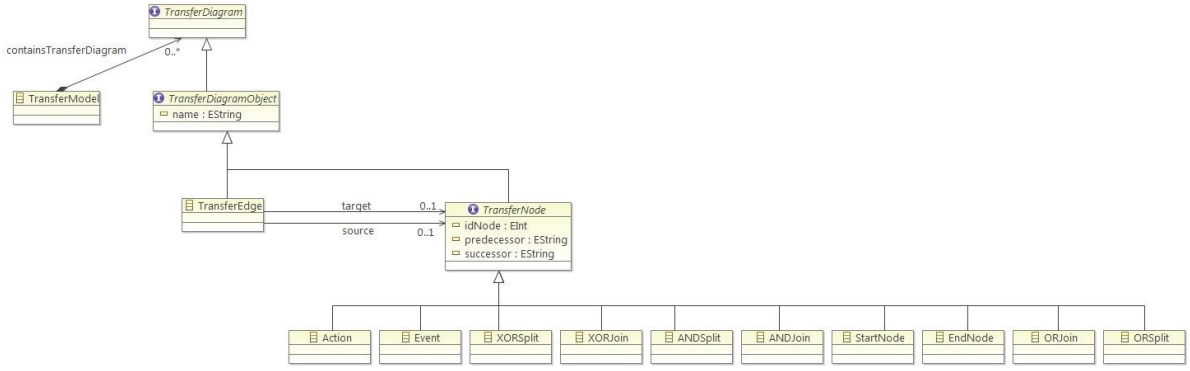
- TransferModel – třída reprezentující diagram
- TransferDiagram – abstraktní třída
- TransferDiagramObject – abstraktní třída, obsahuje atribut **name:EString**
- TransferEdge
- TransferNode – abstraktní třída, obsahuje atribut **idNode:EInt, predecessor:EString, successor:EString**
- Action
- Event
- XORSplit
- XORJoin
- ANDSplit
- ANDJoin
- ORSplit
- ORJoin

Agregace mezi entitami:

- containsTransferDiagram

Asociace mezi entitami:

- source
- target



Obr. 25: Transformační mezi-metamodel

4.6 Definice grafických prvků

Grafický model definuje jednotlivé prvky pro notace byznys procesů. V následujících tabulkách jsou zobrazeny grafické prvky, jak byly navrženy v rámci vývoje notací na platformě EMF.

4.6.1 EPC

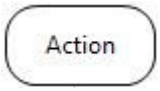
Tabulka určuje, jak jsou graficky navrženy jednotlivé prvky EPC. Grafické znázornění respektuje pravidla a specifikace dané metodiky.

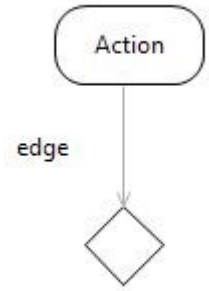
Symbol	Popis
	Aktivita
	Událost
	Logická spojka AND
	Logická spojka OR
	Logická spojka XOR
	Přechod

Tab. 7: grafické prvky EPC

4.6.2 Diagram Aktivit

Tabulka určuje, jak jsou graficky navrženy jednotlivé prvky diagramu aktivit. Grafické znázornění respektuje pravidla a specifikace dané metodiky.

Symbol	Popis
	Aktivita
	Počáteční stav

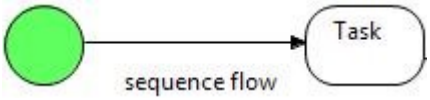
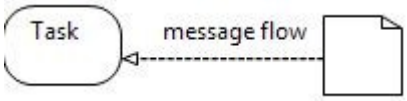
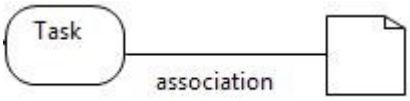
	Konečný stav
	Rozvětvení / Sjednocení
	Rozhodnutí / Sloučení
	Přechod

Tab. 8: grafické prvky Diagramu Aktivit

4.6.3 BPMN

Tabulka určuje, jak jsou graficky navrženy jednotlivé prvky BPMN. Grafické znázornění respektuje pravidla a specifikace dané metodiky.

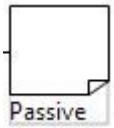
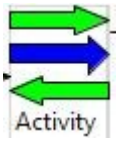
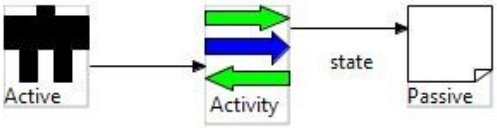
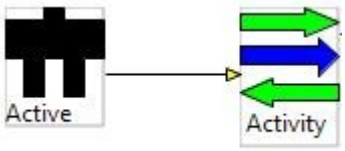
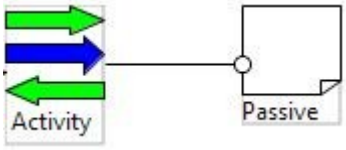
Symbol	Popis
	Aktivita
	Počáteční událost
	Konečný stav
	Událost během procesu

	Datový objekt
	Rozhodovací blok
	Paralelní zpracování
	Sekvenční tok
	Tok zpráv
	asociace
	Bazén
	Plavecká dráha

Tab. 9: grafické prvky BPMN

4.6.4 BP Studio Koordinační Model

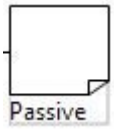
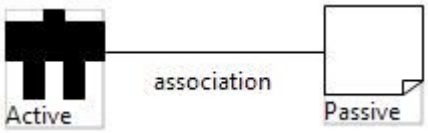
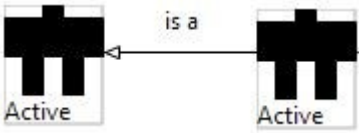
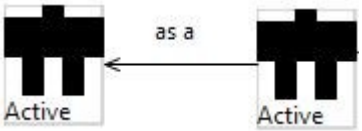
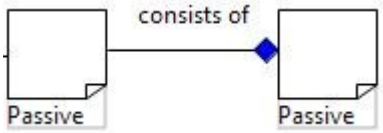
Tabulka určuje, jak jsou graficky navrženy jednotlivé prvky koordinačního modelu BP Studia. Grafické znázornění respektuje pravidla a specifikace dané metodiky.

Symbol	Popis
	Aktivní objekt
	Pasivní objekt
	Proces
	Activita
	Tok
	Zodpovědnost
	Inhibice

Tab. 11: grafické prvky BP Studio Koordinační Model

4.6.5 BP Studio Objektový Model

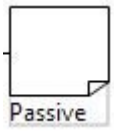
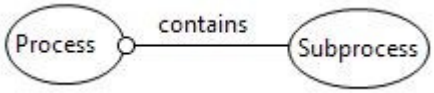
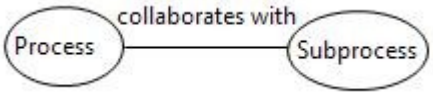
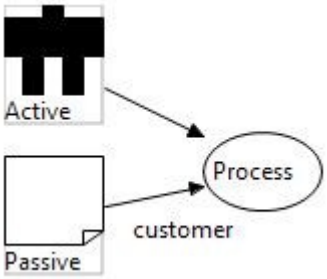
Tabulka určuje, jak jsou graficky navrženy jednotlivé prvky objektového modelu BP Studia. Grafické znázornění respektuje pravidla a specifikace dané metodiky.

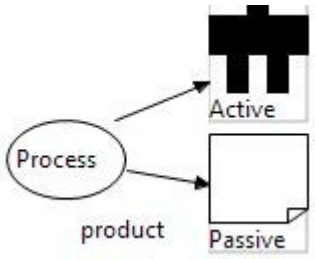
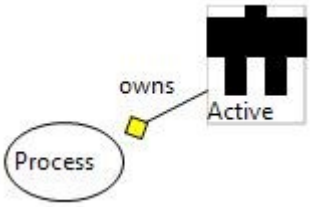
Symbol	Popis
	Aktivní objekt
	Pasivní objekt
	Asociace
	Zobecnění
	Role
	Kompozice

Tab. 12: grafické prvky BP Studio Objektový Model

4.6.6 BP Studio Funkční Model

Tabulka určuje, jak jsou graficky navrženy jednotlivé prvky funkčního modelu BP Studia. Grafické znázornění respektuje pravidla a specifikace dané metodiky.

Symbol	Popis
	Aktivní objekt
	Pasivní objekt
	Proces
	Obsažení
	Spolupráce
	Zákazník

	Produkt
	Vlastník

Tab. 13: grafické prvky BP Studio Funkční Model

4.7 Pluginy

Pluginy jsou vygenerovány pro jednotlivé notace. Pro spuštění v prostředí Eclipse viz. kapitola 4.2, je potřeby nakopírovat všechny pluginy do složky …\eclipse\plugins\. Poté jsou modely a diagramy připraveny k použití. Pluginy jsou rozděleny podle notací a každá má čtyři vlastní pluginy:

- Model – vytváří model ve stromové struktuře.
- Diagram – vytváří diagram, který je odvozen z modelu a umožňuje modelování grafických prvků.
- Editor – editor reprezentuje funkce a operace dostupné pro model, tento plugin integruje nástroje pro transformace, import a export modelů.
- Edit – provider poskytující propojení mezi editorem a model.

4.8 Praktické využití

V následující kapitole se budeme věnovat praktickému využití implementace notací na platformu EMF. V době kdy se stále více firmy zaměřují na procesy a také sledování byznys procesů v rámci implementace různých systémů. Námi implementovaná práce umožňuje využít pluginy jednotlivých notací k modelování procesů na otevřené a volně dostupné platformě Eclipse. Součástí funkcionality pluginů jsou také transformace jednotlivých modelů, což může být přínosem v různých fázích implementace systému ve firmě, který ve své podstatě odráží procesy firmy ve své funkcionalitě. Tyto modely mohou být nejen přínosné pro případné implementátory, ale také pro komunikaci se zákazníkem nad problematikou firmy. Každý takovýto zákazník je jiný a převody mezi jednotlivými notacemi modelů, mohou také napomoci k lepšímu porozumění si se zákazníkem u modelovaných procesů.

4.8.1 Otestování pluginů

V po implementační fázi naší práce jsme otestovali funkčnost a využitelnost. Součástí diplomové práce jsou také vzorové diagramy a modely, na kterých jsme testovali funkce modelování a transformací mezi jednotlivými notacemi. Diagramy umožňují také import a export do XML souborů pro snadnou přenositelnost modelů a následovnou možnost integrace s jinými nástroji. Diagramy a modely notací jsme testovali na malých i středně velkých model a nevyskytli se problémy při modelování ani v rámci transformací, tudíž jsou jednotlivé pluginy připraveny k použití v běžné praxi.

4.8.2 Možnosti využití

Mezi hlavní možnosti využití jsou modelovací a transformační schopnosti pluginů, například pro menší firmy, které z důvodů investic do vývojových a analytických nástrojů zvolí platformu Eclipse. Pluginy notací byznys procesů jsou použitelné jak pro analytické zpracování byznys procesů ve firmě, tak v různých částech implementace systémů, kdy potřebujeme dokumentovat či specifikovat procesy a části systémů. Importní a exportní funkce pluginů nám také umožní vytvořit integrační rozhraní i do jiných nástrojů, jelikož výstupem z pluginu jsou formalizované XML soubory.

5 Závěr

Cílem páce bylo implementovat BP Studio modely a modely dalších notací na platformu EMF. V rámci implementace dále vyvinout nástroje pro transformace, import a export mezi jednotlivými notacemi. Tyto funkce mohou být spolu s grafickými nástroji pro modelování nápomocné například v lepší komunikaci mezi analytiky zabývající se byznys procesy. Výsledná implementace byla vygenerována do pluginů, které umožňují snadné nasazení nástrojů do rozhraní Eclipse.

Jako první krok bylo potřeba definovat metamodely jednotlivých notací na platformě EMF. K tomu bylo zapotřebí porozumět a naučit se definovat modely v metamodelu EMF zvaném Ecore. Tento Ecore Model je základním kamenem pro budování celých pluginů. Z implementace Ecore metamodelů vyplynula nutnost poznání další platformy GMF. Za pomoci GMF byly navrhnuty a implementovány nástroje pro grafické modelování byznys procesů. V rámci vývoje funkcí transformace, byl tak vyvinut transformační mezi-metamodel, který byl poté převeden také na platformu EMF. Při integraci transformačních, importních a exportních nástrojů, byl také důkladně poznán generovaný kód, který nabízí širokou škálu „customizace“.

Při implementaci byly poznány možnosti dalšího rozšíření pluginů přes generovaný kód EMF a GMF, umožňující širokou škálovatelnost kódu a podporující vysokou úroveň modifikace. Pro jednotlivé modely a diagramy lze dodat například funkcionalitu zachycující chování elementů, provádět požadované validace nad strukturovanými modely či generovat další modely v závislosti na požadavcích uživatele.

6 Literatura

[1] VAN GIGCH, John P. System design modeling and metamodeling. New York: Plenum Press, c1991, 453 s. ISBN 03-064-3740-6.

[2] BPMN 2.0. [online]. [cit. 2012-05-03]. Dostupné z: http://www.omg.org/spec/BPMN/2.0/

[3] ARLOW, Jim, NEUSTADT, Ila. UML 2 a unifikovaný proces vývoje aplikací. Brno : Computer Press, 2007. 567 s. ISBN 978-80-251-1503-9.

[4] BP Studio. [online]. [cit. 2012-05-03]. Dostupné z: http://vondrak.cs.vsb.cz/download/bpstudio_eval/manual.pdf

[5] STEINBERG, Dave. EMF: eclipse modeling framework. 2nd ed. Upper Saddle River: Addison-Wesley, c2009, 704 s. ISBN 978-0-321-33188-5.

[6] Essential EMF. [online]. [cit. 2012-05-03]. Dostupné z: http://refcardz.dzone.com/refcardz/essential-emf

[7] EMF Tutorial. [online]. [cit. 2012-05-03]. Dostupné z: http://www.vogella.com/articles/EclipseEMF/article.html

[8] MOORE, Bill. Eclipse development using the graphical editing framework and the eclipse modeling framework. 1st ed. Research Triangle Park, NC: IBM, International Technical Support Organization, 2004, 238 s. ISBN 07-384-5316-1.

[9] GMF Tutorial. [online]. [cit. 2012-05-03]. Dostupné z: http://wiki.eclipse.org/index.php/GMF_Tutorial

[10] GMF Tutorial BPMN. [online]. [cit. 2012-05-03]. Dostupné z: http://wiki.eclipse.org/GMF_Tutorial_BPMN

[11] Meta Modeling With Eclipse. [online]. [cit. 2012-05-03]. Dostupné z: smv.unige.ch//research-projects/mtv/files/MetaModelingWithEclipse.pdf

[12] Metody Byznys Modelování. [online]. [cit. 2012-05-03]. Dostupné z: vondrak.cs.vsb.cz/download/Metody_byznys_modelovani.pdf

Příloha A

Instalace Pluginů

Eclipse

Pro instalaci Eclipse je výběr ze dvou možností:

- Eclipse Classic 3.7.2
- Eclipse Modeling Tools - Version: Indigo Service Release 2

Instalační zdroje je možné stáhnout z http://www.eclipse.org/downloads/.

Instalované komponenty

Po instalaci IDE Eclipse je potřeba také doinstalovat patřičné komponent. V Eclipse zadejte *Help > Install New Software*.

URL adresa pro stahování komponent: Indigo - http://download.eclipse.org/releases/indigo.

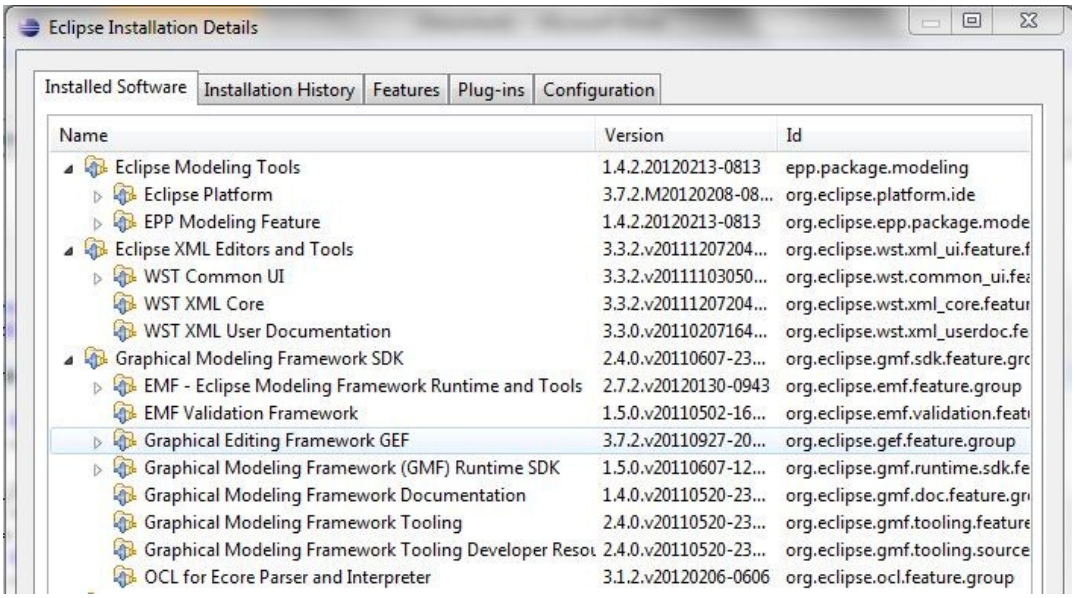
Pro Eclipse Classic:

- Eclipse Modeling Framework (EMF)
- Graphical Modeling Framework (GMF)

Pro Eclipse Modeling Tools

- Graphical Modeling Framework (GMF)

Pro překontrolování nainstalovaných součástí v Eclipse zvolte Help > About Eclipse > Installation Detais záložka Installed Software (viz. Obr. 38).



Obr. 26: Eclipse Installation Details

Pluginy

Pro instalaci pluginů do prostředí Eclipse je nutné nakopírovat všechny pluginy do složky …\eclipse\plugins\.

Pluginy jsou součástí přiloženého CD k diplomové práci ve složce Plugins.

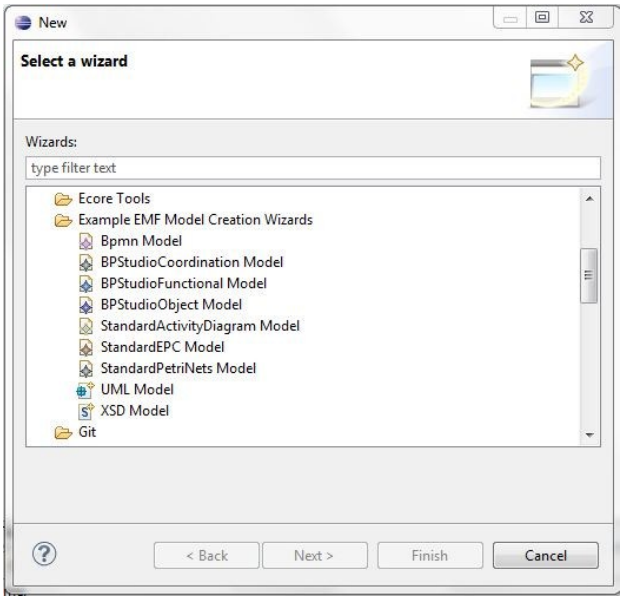
Práce s diagramy a modely

Modely jsou určeny pro práci se stromovou strukturou notace a přes tyto objekty je možné spouštět transformace, import a export. Diagramy slouží ke grafické specifikaci modelu. Model a diagram jsou navzájem propojeny a při vytvoření elementu v jednom objektu se převede element do objektu druhého.

Vytvoření modelu

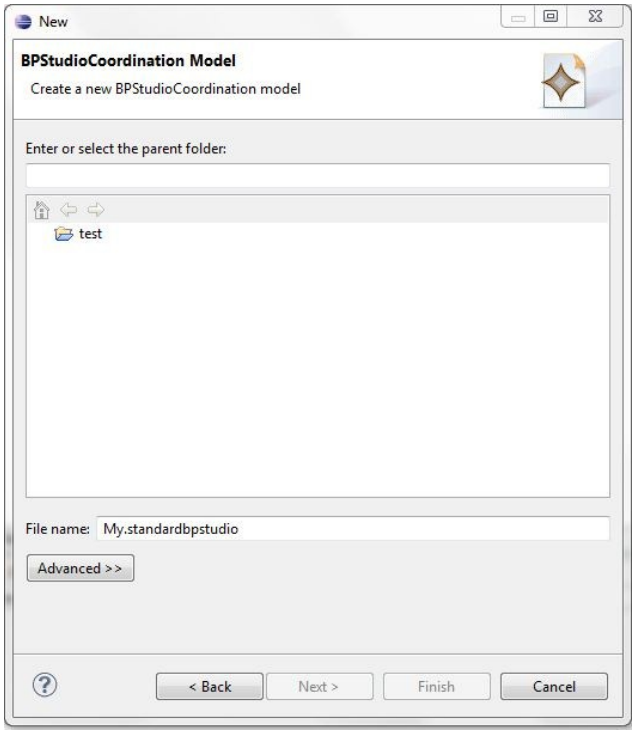
- Vytvořit nový projekt New > Project > General > Project.

- Vytvořit nový model New > Other > Example EMF Model Creation Wizards a vybrat daný model.



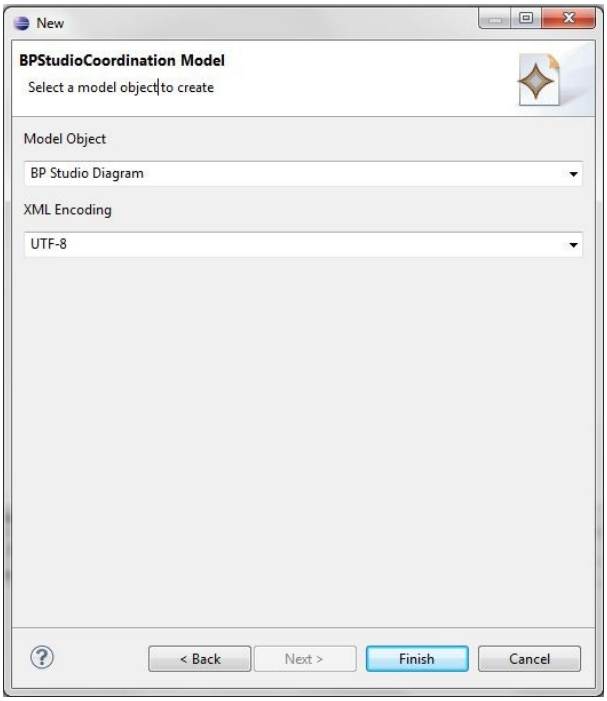
Obr. 27: Průvodce pro vytvoření modelu

- Zadejte jméno a složku pro uložení.



Obr. 28: Průvodce pro vytvoření modelu

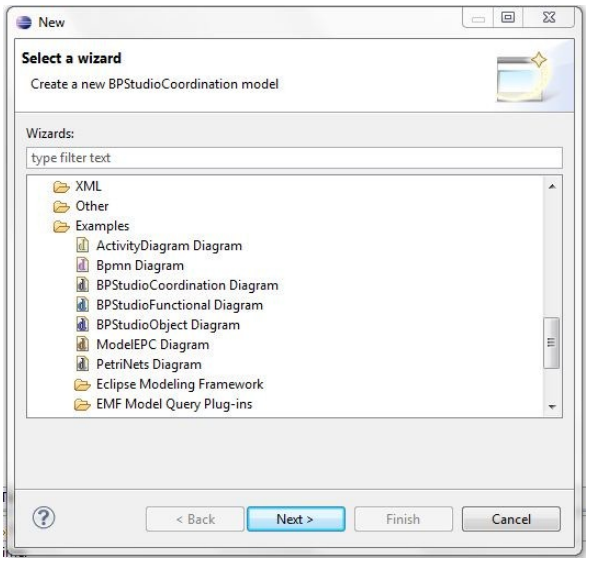
- Zadejte Model Object („jméno modelu“ + Diagram).



Obr. 29: Průvodce pro vytvoření modelu

Vytvoření diagramu

- Vytvořit nový projekt New > Project > General > Project.
- Vytvořit nový model New > Other > Example a vybrat daný diagram.

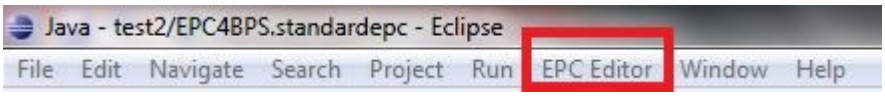


Obr. 30: Průvodce pro vytvoření diagramu

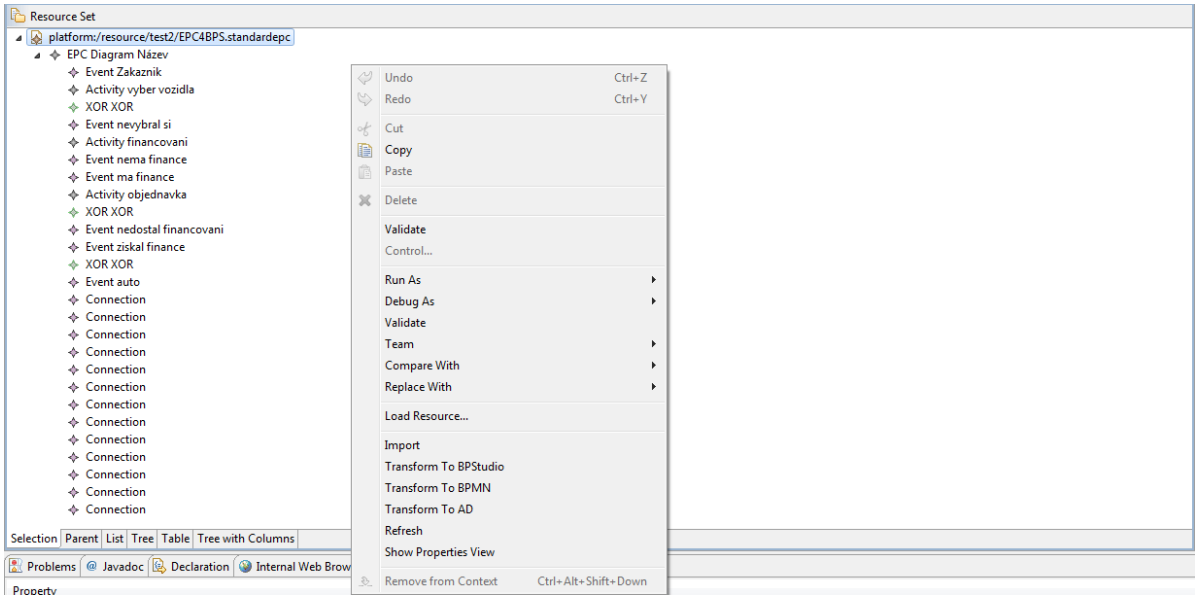
- Zadejte jméno a složku pro uložení při vytvoření diagramu se automaticky vytváří i model, zadejte také jeho jméno a složku pro uložení.

Práce s modelem

- Inicializace diagramu je možná přes pravé tlačítko na model, vybrat operaci **„Initialize *název diagramu* diagram file“**. Po spuštění této operace se vytvoří diagram zadáním jména, složky pro uložení a vybráním „Document root element“.
- Operace dostupné přes panel nástrojů nebo přes pravé tlačítko v oblasti stromové struktury modelu:
    - Transform To BPMN
    - Transform To EPC
    - Transform To AD
    - Transform To BP Studio
    - Import
    - Export



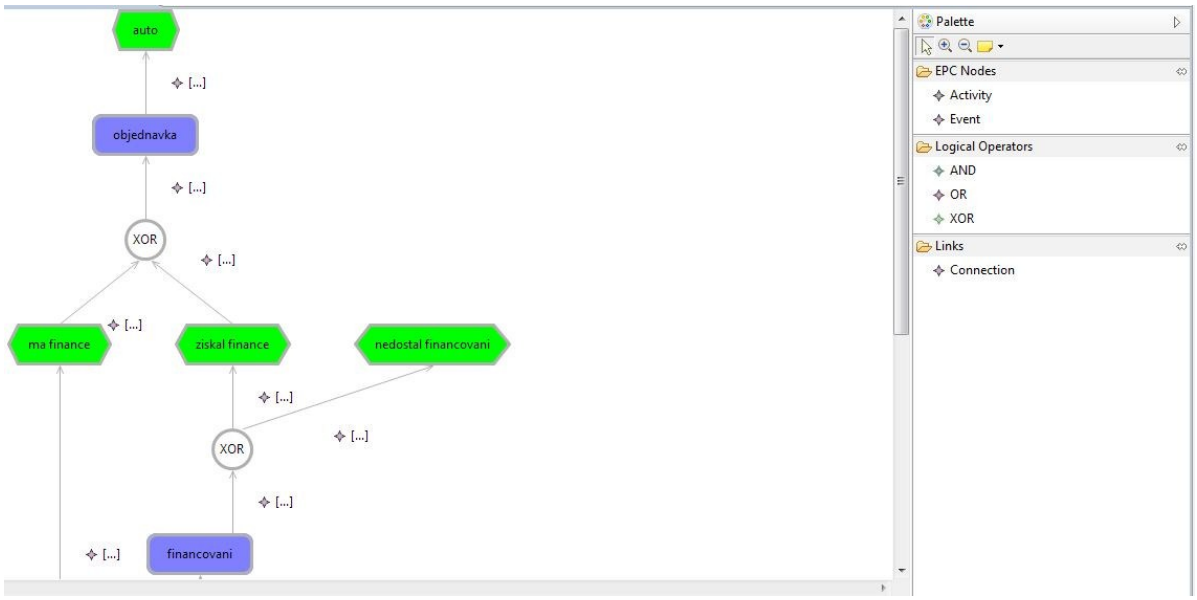
Obr. 31: Menu pro model v hlavním panelu nástrojů Eclipse



Obr. 32: Menu dostupné přes pravé tlačítko na model

Práce s diagramem

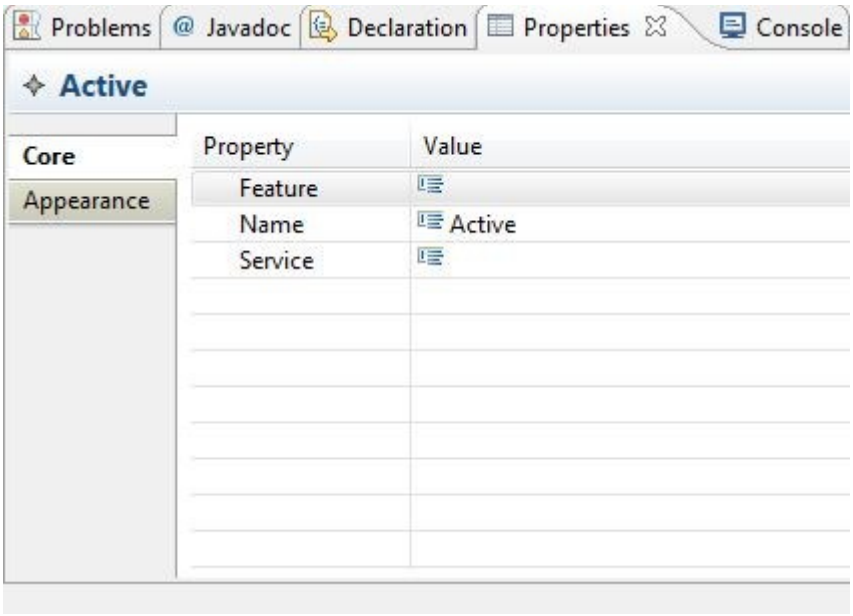
Pomocí palety je možné vybírat elementy a spojení pro modelování diagramu. Po uložení se automaticky namodelované objekty ukládají do příslušného modelu diagramu.



Obr. 33: Pracovní okno pro modelování diagramu

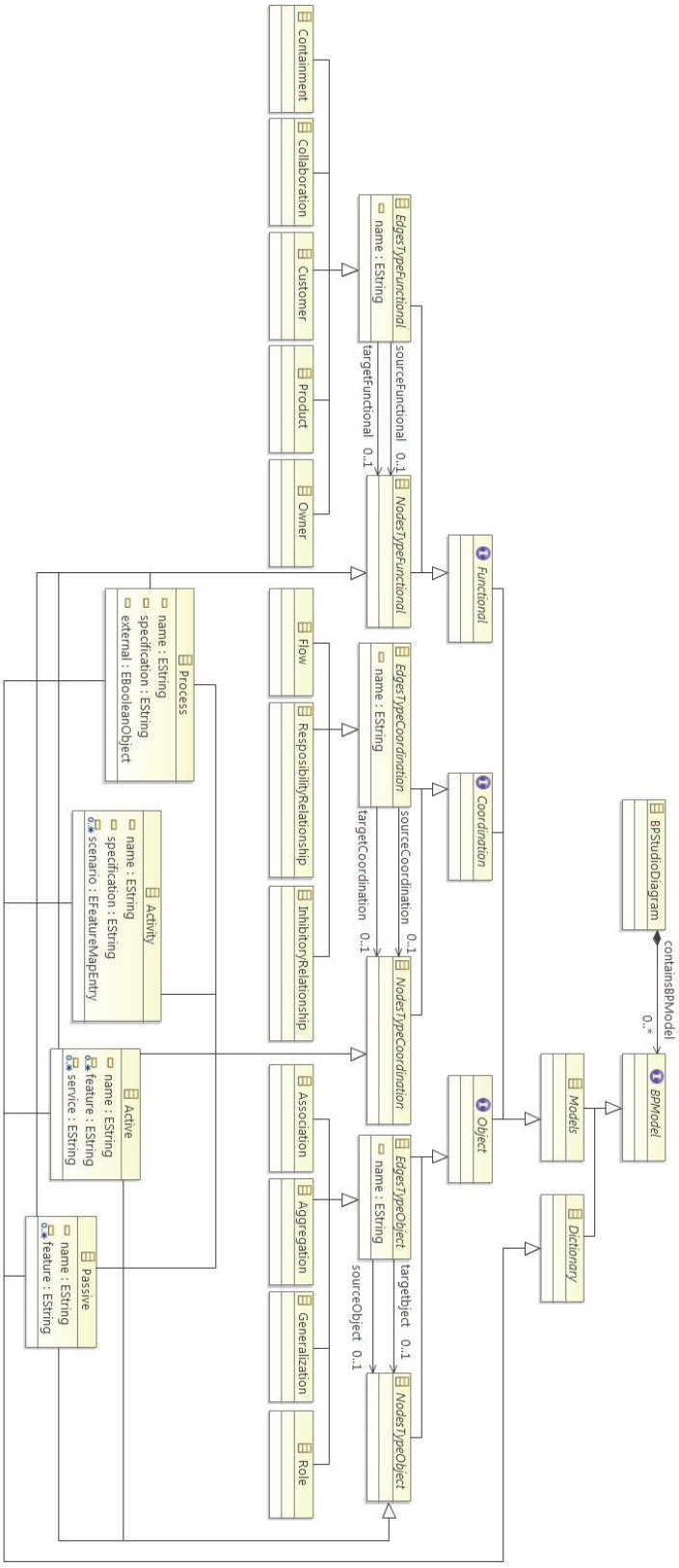
Vlastnosti objektů diagramu a modelu

Jednotlivé objekty jsou definovány sadou vlastností vyplívajících z metamodelů notací, např. name, feature, service, id, weight. Jednotlivé vlastnosti jsou editovatelné přímo v diagramu jako například name nebo jsou dostupné přes záložku „Properties“ v Eclipse. Pro zobrazení záložky „Properties“ jděte na hlavní panel nástrojů Eclipse a Window > Show View > Properties.



Obr. 34: Properties v Eclipse

Příloha B



Obr. 35: Metamodel BP Studia

Příloha C

Obsah přiloženého CD:

- **Diplomová práce**

  Text diplomové práce doc/pdf.

- **Plugins**

  Pluginy pro nakopírování a spuštění v prostředí Eclipse.

- **Vzorové soubory**

  Vzorové soubory pro transformace, import a export.

- **Zdrojové kódy**

  Zdrojové kódy pluginů notací.